PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G10L 3/00

A1

(11) International Publication Number: WO 96/32710

(43) International Publication Date: 17 October 1996 (17.10.96)

(21) International Application Number: PCT/US96/04974

(22) International Filing Date: 10 April 1996 (10.04.96)

(30) Priority Data:

08/419,200	10 April 1995 (10.04.95)	US
08/420,721	10 April 1995 (10.04.95)	US

(71) Applicant (*for all designated States except US*): CORPORATE COMPUTER SYSTEMS, INC. [US/US]; Building #4, 670 North Beers Road, Holmdel, NJ 07733 (US).

(72) Inventor; and

(75) Inventor/Applicant (*for US only*): HINDERKS, Larry, W. [US/US]; 37 Ladwood Drive, Holmdel, NJ 07733 (US).

(74) Agents: SMALL, Dean, D. et al.; McAndrews, Held & Malloy, Ltd., Suite 3400, 500 West Madison, Chicago, IL 60661 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, US, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

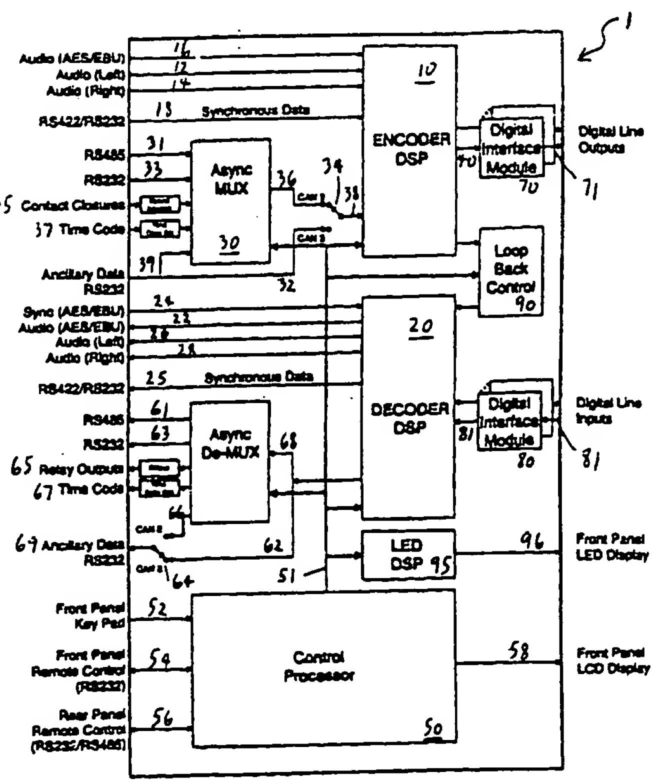
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: SYSTEM FOR COMPRESSION AND DECOMPRESSION OF AUDIO SIGNALS FOR DIGITAL TRANSMISSION

(57) Abstract

Ancillary data (39) may be multiplexed (30) and encoded (10) with audio data (12, 14) and transmitted (70, 71) in such a way that it may be decoded (20) when received (80, 81).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia
AT	Austria
AU	Australia
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CS	Czechoslovakia
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgystan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakhstan
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam

SYSTEM FOR COMPRESSION AND DECOMPRESSION OF AUDIO SIGNALS FOR DIGITAL TRANSMISSION

RELATED APPLICATION

The present application relates to co-pending PCT application ________________, filed April 10, 1996, entitled "Method and Apparatus for Transmitting Coded Audio Signals Through a Transmission Channel With Limited Bandwidth" by the same inventor and assigned to the Assignee of the present application. The co-pending PCT application noted above is incorporated by reference in its entirety along with any appendices and attachments thereto.

SOURCE CODE APPENDIX

Source code for the control processor of the present invention has been included as a SOURCE CODE APPENDIX.

FIELD OF THE INVENTION

The present invention relates generally to an audio CODEC for the compression and decompression of audio signals for transmission over digital facilities, and more specifically, relates to an audio CODEC that is programmable by a user to control various CODEC operations, such as monitoring and adjusting a set of psycho-acoustic parameters, selecting different modes of digital transmission, and downloading new compression algorithms.

BACKGROUND OF THE INVENTION

Current technology permits the translation of analog audio signals into a sequence of binary numbers (digital). These numbers may then be transmitted and received through a variety of means. The received signals may then be converted back into analog audio signals. The device for performing both the conversion from analog to digital and the conversion from digital to analog is called a CODEC. This is an acronym for COder/DECoder.

The cost of transmitting bits from one location to another is a function of the number of bits transmitted per second. The higher the bit transfer rate the higher the cost. Certain laws of physics in human and audio perception establish a direct relationship between perceived audio quality and the number of bits transferred per second. The net result is that improved audio quality increases the cost of transmission.

CODEC manufacturers have developed technologies to reduce the number of bits required to transmit any given audio signal (compression techniques) thereby reducing the associated transmission costs. The cost of transmitting bits is also a function of the transmission facility used, i.e., satellite, PCM phone lines, ISDN (fiber optics).

A CODEC that contains some of these compression techniques also acts as a computing device. It inputs the analog audio signal, converts the audio signal to a digital bit stream, and then applies a compression technique to the bit stream thereby reducing

the number of bits required to successfully transmit the original audio signal. The receiving CODEC applies the same compression techniques in reverse (decompression) so that it is able to convert the compressed digital bit stream back into an analog audio signal. The difference in quality between the analog audio input and the reconstructed audio output is an indication of the quality of the compression technique. The highest quality technique would yield an identical signal reconstruction.

Currently, the most successful compression techniques are called perceptual coding techniques. These types of compression techniques attempt to model the human ear. These compression techniques are based on the recognition that much of what is given to the human ear is discarded because of the characteristics of the ear. For example, if a loud sound is presented to a human ear along with a softer sound, the ear will only hear the loud sound. As a result, encoding compression techniques can effectively ignore the softer sound and not assign any bits to its transmission and reproduction under the assumption that a human listener can not hear the softer sound even if it is faithfully transmitted and reproduced.

Many conventional CODECs use perceptual coding techniques which utilize a basic set of parameters which determine their behavior. For example, the coding technique must determine how soft a sound must be relative to a louder sound in order to make the softer sound a candidate for exclusion from transmission. A

number which determines this threshold is considered a parameter of the scheme which is based on that threshold. These parameters are largely based on the human psychology of perception so they are collectively known as psycho-acoustic parameters.

However, conventional CODECs which use perceptual coding have experienced limitations. More specifically, manufacturers of existing CODECs preprogram all of the CODEC's operating variables which control the compression technique, decompression technique, bit allocation and transmission rate. By preprogramming the CODEC, the manufacturer undesirable limits the user interaction with the resulting CODEC. For example, it is known that audio can be transmitted by digital transmission facilities. These digital transmissions include digital data services, such as conventional phone lines, ISDN, T1, and E1. Other digital transmission paths include RF transmission facilities such as spread spectrum RF transmission and satellite links.

Although existing CODECs can transmit compressed audio signals via digital transmission facilities, any variables regarding the mode of transmission are preprogrammed by the manufacturer of the CODEC, thereby limiting the CODEC's use to a single specific transmission facility. Hence, the user must select a CODEC which is preprogrammed to be compatible with the user's transmission facility. Moreover, existing CODECs operate based on inflexible compression and bit allocation techniques and thus, do not provide users with a method or apparatus to monitor or adjust the CODEC to

fit the particular user's wants and needs. Accordingly, users must test CODECs with different compression and bit allocation techniques and then select the one device which has the features or options so desired, e.g. satellite transmission capabilities.

Moreover, standard coding techniques have been developed in order to ensure interoperability of CODECs from different manufacturers and to ensure an overall level of audio quality, thereby limiting the CODEC's use to a single specific transmission facility. One such standard is the so-called ISO/MPEG Layer-II compression standard, for the compression and decompression of an audio input. This standard sets forth a compression technique and a bit stream syntax for the transmission of compressed binary data. The ISO/MPEG Layer-II standard defines a set of psycho-acoustic parameters that is useful in performing compression. U.S. Patent No. 4,972,484, entitled "Method of Transmitting or Storing Masked Sub-band Coded Audio Signals," discloses the ISO/MPEG Layer-II standard and is incorporated by reference.

However, conventional CODECs do not use a uniform set of parameters. Each CODEC manufacturer determines their own set of psycho-acoustic parameters either from a known standard or as modified by the manufacturer in an attempt to provide the highest quality sound while using the lowest number of bits to encode audio. Once the manufacturer selects a desired parameter set, the manufacturer programs values for each of the parameters. These

preprogrammed parameter values correspond to the manufacturer's perception of an optimal audio quality at the decoder.

However, in conventional CODECs, users typically are unaware of the existence or nature of these parameters. Further, the user has no control over the parameter values. As a result, users were required to test different CODECs from different manufacturers and then select the CODEC that met the user's requirements or that sounded best to the user.

Typically, conventional CODECs utilize standard parameters which have been accepted by the International Standards Organization (ISO) and have been adopted as part of the International Standards Organization, Motion Picture Experts Group (ISO/MPEG) Layer-II compression standard. However, the ISO/MPEG Layer-II standard has met with limited acceptance since these parameters do not necessarily provide CD quality output. The ISO/MPEG Layer-II parameters are determined and set based on the average human ear. The parameters do not account for the variations between each individual's hearing capabilities. Hence, the conventional standards and CODECs do not afford the ability for users to tune their CODEC to the user's individual subjective hearing criteria. Nor are conventional CODECs able to meet changing audio needs and to shape the overall sound of their application.

A need remains within the industry for an improved CODEC which is more flexible, programmable by the user, and which overcomes the

disadvantages experienced heretofore. It is an object of the present invention to meet this need.

OBJECTS OF THE INVENTION

It is an object of the present invention to provide a programmable audio CODEC that can be monitored, controlled and adjusted by a user to control the various functions of the CODEC.

It is another object of the present invention to provide an audio CODEC that is programmable by a user to transmit compressed digital bit streams over various user selected digital transmission facilities.

It is an object of the present invention to provide a user programmable audio CODEC with a plurality of psycho-acoustic parameters that can be monitored, controlled, and adjusted by a user to change the audio output from the CODEC.

It is a related object of the present invention to provide an audio CODEC with new psycho-acoustic parameters.

It is a further related object of the present invention to provide an audio CODEC where the psycho-acoustic parameters are changed by knobs on the front panel of the CODEC.

It is another related object of the present invention to provide an audio CODEC where the psycho-acoustic parameters are changed by a keypad on the front panel of the CODEC.

It is still a further related object of the present invention to provide an audio CODEC with a personal computer connected

thereto to adjust the psycho-acoustic parameters by changing graphic representations of the parameters on a computer screen.

It is a related object of the present invention to provide an audio CODEC that is programmable by a user to transmit compressed digital bit streams over a digital data service.

It is a further related object of the present invention to provide an audio CODEC that is programmable by a user for transmission of compressed digital bit streams over any of T1, E1 and ISDN lines or over RF transmission facilities.

It is yet another related object of the present invention to provide an audio CODEC that is user programmable for transmission of compressed digital bit streams via satellite.

It is a further object of the present invention to provide an audio CODEC for transmission of asynchronous data together with the transmission of compressed audio.

It is still a further object of the present invention to provide an audio CODEC that utilizes the multiple audio compression and decompression schemes.

It is still another object of the present invention to provide an audio CODEC which allows a user to select one of several stored audio compression techniques.

It is still another object of the present invention to provide an audio CODEC that is remotely controlled by a host computer.

It is still another object of the present invention to provide an audio CODEC for monitoring either the encoder input signal or the decoder output signal with the use of headphones.

It is still another object of the present invention to provide an audio CODEC with safeguards for automatically selecting a second transmission facility if a first user selected transmission facility fails.

It is yet another object of the present invention to provide an audio CODEC that can be controlled by inputting control commands into a key pad on the front panel of the CODEC.

It is related object of the present invention to provide an audio CODEC having a user interface to control and program the audio CODEC through the use of a graphics display on the front panel.

It is still another related object of the present invention to provide for connection of a personal computer to the audio CODEC for controlling the input of program information thereto.

It is still another object of the present invention to provide bi-directional communication between two audio CODECs.

It is still another object of the present invention to provide an audio CODEC that can be interfaced to a local area network.

It is yet another object of the present invention to provide an audio CODEC that will provide programmed information to users through the use of indicators on the front panel of the CODEC.

It is yet another object of the present invention to provide an audio CODEC that can send non-audio compressed information including text, video and graphic information.

It is still another object of the present invention to provide an audio CODEC that can store and retrieve information on and from an electronic storage medium or a disk drive.

It is still another related obje~t of the present invention to provide an audio CODEC that can transmit control information along with the textual video and graphic information.

It is still a further object of the present invention to provide digital audio compression techniques that yield improved and preferably CD quality audio.

It is a related object of the present invention to provide a compression scheme that yields better audio quality than the MPEG compression standard.

It is still another related object of the present invention to provide CD quality audio that achieves a 12 to 1 compression ratio.

SUMMARY OF THE INVENTION

The present invention provides a CODEC which holds several compression algorithms and allows the user easily to download future audio compression algorithms as needed. This makes the present CODEC very versatile and prevents it from becoming obsolete.

The preferred CODEC provides for both digital and analog input of external signals. The CODEC is also capable of handling a wide variety of ancillary data which can be incorporated into the compressed bit stream along with the audio and header data. The ancillary bit stream preferably enters the encoder directly from external sources. However, the user could alternatively choose to have the external data multiplexed into a composite ancillary bit stream before being encoded with the audio and header data. The preferred CODEC also provides for rate adaptation of signals that are input (and output) at one rate and compressed (and decompressed) at yet another rate. This rate adaptation can also be synchronized to external clock sources.

The user can also programmably alter the psycho-acoustic compression parameters to optimize transmissions under different conditions. The disclosed invention also allows the user to programmably control CODEC transmission modes as well as other CODEC operations. Such programmable control is achieved through remote interfaces and/or direct keypad control.

The compressed output signal can also be interfaced with a variety of external sources through different types of output Digital Interface Modules (DIMs). Similar input DIMs would input return signals for decoding and decompression by the CODEC. Certain specialized DIMs might also operate as satellite receiver modules. Such modules would preferably store digital information as it becomes available for later editing and use. Satellite

receiver modules would be capable of receiving information such as audio, video, text, and graphics. This information would then be decoded and decompressed as appropriate by the CODEC.

Additional features and advantages of the present invention will become apparent to one of skilled in the art upon consideration of the following detailed description of the present invention.

BRIEF DESCRIPTIONS OF THE DRAWINGS

Figure 1 is a block diagram of a CODEC illustrating signal connections between various components in accordance with a preferred embodiment of the present invention.

Figure 2 is a block diagram of a CODEC illustrating signal connections between various components in accordance with the preferred embodiment shown in Figure 1.

Figure 3 is a block diagram illustrating ancillary data being multiplexed into a composite bit stream in accordance with the preferred embodiment of Figure 1.

Figure 4 is a block diagram illustrating an ISO/MPEG audio bit stream being decoded into a composite ancillary bit stream and audio left and right signals in accordance with the preferred embodiment of Figure 1.

Figure 5 is an example of a front panel user keypad layout in accordance with a preferred embodiment of the present invention.

Figure 6 is another example of a front panel user keypad layout in accordance with a preferred embodiment of the present invention.

Figure 7 is another example of a front panel user keypad layout in accordance with a preferred embodiment of the present invention.

Figure 8 is a block diagram showing the decoder output timing with the AES/EBU sync disabled or not present and using normal timing in accordance with a preferred embodiment of the present invention.

Figure 9 is a block diagram showing the decoder output timing with AES/EBU sync disabled or not present using internal crystal timing in accordance with a preferred embodiment of the present invention.

Figure 10 is a block diagram showing decoder output timing with AES/EBU sync enabled and present using AES timing in accordance with a preferred embodiment of the present invention.

Figure 11 is an example of an LED front panel display in accordance with a preferred embodiment of the present invention.

Figure 12 is another example of an LED front panel display in accordance with a preferred embodiment of the present invention.

Figure 13 is a block diagram of a CODEC illustrating signal connections between various components allowing transmission of audio, video, text, and graphical information in accordance with a preferred embodiment of the present invention.

Figure 14 is a diagram illustrating the interconnection between various modules in accordance with a preferred embodiment.

Figure 15 is a block diagram of an embodiment of an encoder as implemented in the CODEC of the system in accordance with the preferred embodiment shown in Figure 14.

Figure 16 is a diagram illustrating a known representation of a tonal masker as received and recognized by a CODEC system.

Figure 17 is a diagram illustrating a known representation of a tonal masker and its associated masking skirts as recognized by a CODEC system.

Figure 18 is a diagram illustrating a tonal masker and its associated masking skirts as implemented by the encoder of the system in accordance with the preferred embodiment shown in Figure 14.

Figure 19 is a diagram illustrating the representation of the addition of two tonal maskers as implemented by the encoder of the system in accordance with the preferred embodiment shown in Figure 14.

Figure 20 is a block diagram illustrating the adjustment of a single parameter as performed by the encoder of the system in accordance with the preferred embodiment shown in Figure 14.

Figure 21 illustrates a block diagram of an encoder for a single audio channel according to the present invention.

Figure 22 illustrates a data structure used in the preferred embodiment for a frame of data.

Figure 23 illustrates a block diagram of an encoder for two audio channels operated in joint stereo according to the present invention.

Figure 24 illustrates a flow diagram of the process followed by the present invention when adjusting the scaling factors.

Figures 25a and 25b illustrate a flow diagram of the overall process followed by the present invention when assigning encoding levels to the quantizers.

Figure 26 illustrates a flow diagram of the process followed by the present invention when obtaining a global masking threshold.

Figure 27 illustrates a flow diagram of the process followed by the present invention predicting bit allocation for mono, stereo or joint stereo frames.

Figure 28 illustrates a flow diagram of the process followed by the present invention when determining an allocation step for a specific subband.

Figure 29 illustrates a flow diagram of the process followed by the present invention when determining the joint stereo boundary.

Figure 30 illustrates a flow diagram of the process followed by the present invention when assigning a quantization level.

Figure 31 illustrates a flow diagram of the process followed by the present invention when deallocating bits from one or more subbands following the initial allocation process.

Figures 32a and 32b illustrate graphs of exemplary subbands having a portion of the global masking threshold therein and multiple masking-to-noise ratios therein corresponding to multiple allocation steps.

Figure 33 illustrates a deallocation table recorded during bit allocation and de-allocation.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

CODEC System with Adjustable Parameters

With reference to FIGURES 14 and 15, a CODEC 1010 has an encoder 1012 and a decoder 1010. The encoder 1012 receives as input an analog audio source 1016. The analog audio source 1016 is converted by an analog to digital converter 1018 to a digital audio bit stream 1020. The analog to digital converter 1018 can be located before the encoder 1012, but is preferably contained therein. In the encoder 1012, compression techniques compress the digital audio bit stream 1020 to filter out unnecessary and redundant noises. In the preferred embodiment, the compression technique utilizes the parameters defined by the ISO/MPEG Layer-II standard as described in USP 4,972,484, and in a document entitled, "Information Technology Generic Coding Of Moving Pictures And Associated Audio," and is identified by citation ISO 3-11172 Rev. 2. The '484 patent and the ISO 3-11172, Rev. 2 Document are incorporated by reference.

In addition, the compression technique of the preferred embodiment of the encoder 1012 adds several new parameters as explained below. The resultant compressed digital audio bit stream 1022 is then transmitted by various transmission facilities (not shown) to a decoder at another CODEC (not shown). The decoder decompresses the digital audio bit stream and then the digital bit stream is converted to an analog signal.

The compression technique utilized by the CODEC 1010 to compress the digital audio bit stream 1020 is attached as the Source Code Appendix, and is hereby incorporated by reference.

Human Auditory Perception - Generally

The audio compression routine performed by the encoder 1012 is premised on several phenomena of human auditory perception. While those phenomena are generally understood and explained in the ISO Document and '484 patent referenced above, a brief summary is provided hereafter.

Generally, it is understood that when a human ear receives a loud sound and a soft sound, close in time, the human will only perceive the loud sound. In such a case, the loud sound is viewed as "masking" or covering up the quiet or soft sound.

The degree to which the softer sound is masked is dependent, in part, upon the frequencies of the loud and soft sounds and the distance between the frequencies of the loud and soft sounds. For instance, a loud sound at 700 Hz will have a greater masking effect

upon a soft sound at 750 Hz than upon a soft sound at 900 Hz. Further, typically, the ear is more discriminating between loud and soft sounds at low frequencies as compared to loud and soft sounds at high frequencies.

Another aspect of hearing and psycho-acoustics is that a person can hear two tones at the same frequency provided that the softer tone is close enough in amplitude to the louder tone. The maximum difference in amplitude between the two tones of common frequency is referred to as the masking index. The masking index is dependent, in part, upon frequency of the tones. Generally, the masking index increases with frequency. For instance, the masking index of a masking tone at 1000 Hz will be smaller than the masking index of a masking tone at 7000 Hz.

FIGURE 17 illustrates the masking index 1034 for the tonal masker 1024. Thus, the masking effect will be greater for a loud sound at 7000 Hz upon a soft sound 7050 Hz as compared to the masking effect of a loud sound at 700 Hz upon a soft sound at 750 Hz. The masking effect of a sound is defined by its "masking skirt," which is explained below.

The encoder defines maskers and masking skirts based on the above noted masking effects (as explained below in more detail). If masking does occur, then the compression technique will filter out the masked (redundant) sound.

The audio compression technique of the encoder is also premised on the assumption that there are two kinds of sound

maskers. These two types of sound maskers are known as tonal and noise maskers. A tonal masker will arise from audio signals that generate nearly pure, harmonically rich tones or signals. A tonal masker that is pure (extremely clear) will have a narrow bandwidth. The band width of a tonal masker varies with frequency. In particular, tones at high frequency may have a wider bandwidth than low frequency tones. For instance, a sound centered at 200 Hz with a width of 50 Hz may not be considered a tone, while a sound centered at 7000 Hz with a width of 200 Hz could be considered a tone. Many sounds have no single dominant frequency (tonal), but instead are more "noise" like. If a sound is wide in bandwidth, with respect to its center frequency, then the sound is classified as noise and may give rise to a noise masker. A noise masker will arise from signals that are not pure. Because noise maskers are not pure, they have a wider bandwidth and appear in many frequencies and will mask more than the tonal masker.

FIGURE 16 illustrates a tonal masker 1024 as a single vertical line at a frequency which remains constant as the power increases to the peak power 1026. By way of example, the tonal masker may have 46 HZ bandwidth. Sounds within that bandwidth, but below the peak power level 1026 are "masked." An instrument that produces many harmonics, such as a violin or a trumpet, may have many such tonal maskers. The method for identifying tonal maskers and noise maskers is described in the ISO Document and the '484 patent referenced above.

FIGURE 17 shows a tonal masker 1024 with its associated masking skirt 1028. The masking skirt 1028 represents a threshold indicating which signals will be masked by the tonal masker 1024. A signal that falls below the masking skirt 1028 (such as the signal designated 1030) cannot be heard because it is masked by the tone masker 1024. On the other hand, a smaller amplitude tone (such as tone 1032) can be heard because its amplitude rises above the masking skirt 1028.

As shown in FIGURE 17, the closer in frequency a signal is to the tonal masker 1024, the greater its amplitude may be and still be masked. Signals that have very different frequencies from the masker 1024, such as signal 1032, may have a lower amplitude and not fall below the masking skirt 1028, nor be masked.

Another aspect of hearing and psycho-acoustics is that a person can hear two tones at the same frequency provided that the softer tone is close enough in amplitude to the louder tone. The maximum difference in amplitude between the two tones of common frequency is referred to as the masking index. The masking index is dependent, in part, upon frequency of the tones. Generally, the masking index increases with frequency. For instance, the masking index of a masking tone at 1000 Hz will be smaller than the masking index of a masking tone at 7000 Hz.

FIGURE 17 illustrates the masking index 1034 for the tonal masker 1024. The masking index 1034 is the distance from the peak 1026 of the tonal masker 1024 to the top 1036 of the masking skirt

1028. This distance is measured in dB. For purposes of illustration, the graphs in FIGURES 16-19 scale the frequency along the modules of the graph in Bark. Each Bark corresponds to a frequency band distinguished by the human auditory system (also referred to as a "critical band"). The human ear divides the discernable frequency range into 24 critical bands. The frequency in psycho-acoustics is often measured in Bark instead of Hertz. There is a simple function that relates Bark to Hertz. The frequency range of 0 to 20,000 Hertz is mapped nonlinearly onto a range of approximately 0 to 24 Bark, according to a known function.

At low frequencies, the human ear/brain has the ability to discern small differences in the frequency of a signal if its frequency is changed. As the frequency of a signal is increased, the ability of the human ear to discern differences between two signals with different frequencies diminishes. At high frequencies, a signal must change by a large value before the human auditory system can discern the change.

As noted above, signals which lack a dominant frequency may be produce noise maskers. A noise masker is constructed by summing all of the audio energy within 1 Bark (a critical band) and forming a single discrete "noise" masker at the center of the critical band. Since there are 24 Bark (critical bands) then there are 24 noise maskers. The noise maskers are treated just like the tonal maskers. This means that they have a masking index and a masking

skirt. It is known that an audio signal may or may not have tonal maskers 1024, but it will generally have 1024 noise maskers.

FIGURE 18 illustrates a masking skirt 1029 similar to that described in the ISO/MPEG Layer-II for psycho-acoustic model I. The masking skirt 1029 is more complex than that of FIGURE 17. The masking skirt 1029 includes four mask portions 1050, 1052, 1054, and 1056, each of which has a different slope. The mask portions 1052-1056 are defined by the following equations:

1) Skirt Portion 1050 = E;

2) Skirt Portion 1052 = F * P+G;

3) Skirt Portion 1054 = H; and

4) Skirt Portion 1056 = I - J*P,

wherein the variables E, F, G, H, I and J represent psycho-acoustic parameters which are initially defined in preset tables, but may be adjusted by the user as explained below. The variable P represents the amplitude 1027 of the masker 1025 to which the masking skirt 1029 corresponds. Thus, the slopes of the mask portions 1050-1056 depend on the amplitude P of the masker 1025. The distance DZ, indicated by the number 1053, represents the distance from the masker 1025 to the signal being masked. As the distance DZ increased between the masker 1029 and the signal to be masked, the masker 1029 is only able to cover up lower and lower amplitude signals. The masking index, AV, indicated by the number 1055, is a function of the frequency. The masking index 1055 for tonal and noise maskers are calculated based on the following formula:

5) $AV_{Tonal} = A + B*Z$; and

6) $AV_{Noise} = C + D*Z$;

wherein the variables A, B, C and D represent psycho-acoustic parameters and the variable Z represents the frequency of the masker in Bark. The parameters A-J and suggested values therefor have been determined by readily available psycho-acoustic studies. A summary of such studies is contained in the book by Zweiker and Fastl entitled "Psychoacoustics," which is incorporated herein by reference.

ISO/MPEG LAYER-II

The CODEC 1010 utilizes the psycho-acoustical model as described in the ISO psycho-acoustical model I as the basis for its parameters. The ISO model I has set standard values for ten model parameters (A, B, ... J). These model parameters are described below:

A	=	6.025	dB
B	=	0.275	dB/Bark
C	=	2.025	dB
D	=	0.175	dB/Bark
E	=	17.0	dB/Bark
F	=	0.4	1/Bark
G	=	6.0	dB/Bark
H	=	17.0	dB/Bark
I	=	17.0	dB/Bark
J	=	.15	1/Bark

Parameters A through J are determined as follows:

Z = freq in Bark

DZ = distance in Bark from master peak (may be + or -) as shown in FIGURE 5

Pxx(Z(k)) = Power in SPL(96 db = +/-32767) at frequency Z of masker K

xx = tm for tonal masker or nm for noise masker

Pxx is adjusted so that a full scale sine wave (+/-32767) generates a Pxx of 96 db.

Pxx = XFFT + 96.0 where XFFT = 0 db at +/-32767 amplitude

XFFT is the raw output of an FFT. It must be scaled to convert it to Pxx

AVtm(k) = A + B * Z(k) Masking index for tonal masker k

AVnm(k) = C + D * Z(k) Masking index for tonal masker k

VF(k,DZ) = E * (|DZ| - 1) + (F * X(Z(k)) + G)

VF(k,DZ) = (F * X(Z(k)) + G) * |DZ|

VF(k,DZ) = H * DZ

VF(k,DZ) = (DZ - 1) * (I - J * X(Z(k))) + H

MLxx(k,DZ) = Pxx(k) - (AVxx(K) + VF(k,DZ))

MLxx is the masking level generated by each masker k at a distance DZ from the masker.

where xx = tm or nm
Pxx = Power for tm or nm

Parameters A through J are shown in FIGURE 15. Parameters A through J are fully described in the ISO 11172-3 document.

Additional Parameters Added to ISO/MPEG LAYER-II

In addition to parameters A-J, the CODEC 1010 may use additional parameters K-Z and KK-NN. The CODEC 1010 allows the user to adjust all of the parameters A-Z and KK-NN. The additional parameters K-Z and KK-NN are defined as follows:

Parameter K - joint stereo sub-band minimum value

This parameter ranges from 1 to 31 and represents the minimum sub-band at which the joint stereo is permitted. The ISO specification allows joint stereo to begin at sub-band 4, 8, 12, or 16. Setting K to 5 would set the minimum to 8. Setting this parameter to 1 would set the minimum sub-band for joint stereo to 4.

Parameter L - anti-correlation joint stereo factor

This parameter attempts to determine if there is a sub-band in which the left and right channels have high levels, but when summed together to form mono, the resulting mono mix has very low levels. This occurs when the left and right signals are anti-correlated. If anti-correlation occurs in a sub-band, joint stereo which includes that sub-band cannot be used. In this case, the joint stereo boundary must be raised to a higher sub-band. This will result in greater quantization noise but without the annoyance of the anti-correlation artifact. A low value of L indicates that if there is a very slight amount of anti-correlation, then move the sub-band boundary for joint stereo to a higher valve.

Parameter M - limit sub-bands

This parameter can range from 0 to 31 in steps of 1. It represents the minimum number of sub-bands which receive at least the minimum number of bits. Setting this to 8.3 would insure that sub-bands 0 through 7 would receive the minimum number of bits independent of the psychoacoustic model. It has been found that the psychoacoustic model sometimes determines that no bits are required for a sub-band and using no bits as the model specifies, results in annoying artifacts. This is because the next frame might require bits in the sub-band. This switching effect is very noticeable and annoying. See parameter { for another approach to solving the sub-band switching problem.

Parameter N - demand / constant bit rate

This is a binary parameter. If it is above .499 then the demand bit rate bit allocation mode is requested. If it is below .499 then the fixed rate bit allocation is requested. If the demand bit rate mode is requested, then the demand bit rate is output and can be read by the computer. Also, see parameter R. Operating the CODEC in the demand bit rate mode forces the bits to be allocated exactly as the model requires. The resulting bit rate may be more or less than the number of bits available. When demand bit rate is in effect, then parameter M has no meaning since all possible sub-bands are utilized and the required number of bits are allocated to use all of the sub-bands.

In the constant bit rate mode, the bits are allocated in such a manner that the specified bit rate is achieved. If the model requests less bits than are available, any extra bits are equally distributed to all sub-bands starting with the lower frequency sub-bands.

Parameter O - safety margin

This parameter ranges from -30 to +30 dB. It represents the safety margin added to the psychoacoustic model results. A positive safety margin means that more bits are used than the psychoacoustic model predicts, while a negative safety margin means to use less bits than the psychoacoustic model predicts. If the psychoacoustic model was exact, then this parameter would be set to 0.

Parameter P - joint stereo scale factor mode

This parameter ranges from 0 to .999999. It is only used if joint stereo is required by the current frame. If joint stereo is not needed for the frame, then this parameter is not used. The parameter p is used in the following equation:

$$br = \text{demand bit rate} * p$$

If br is greater than the current bit rate (..128, 192, 256, 384), then the ISO method of selecting scale factors is used. The ISO method reduces temporal resolution and requires less bits. If br is less than the current bit rate, then a special method of choosing the scale factors is invoked. This special model generally requires that more bits are used for the scale factors but it provides a better stereo image and temporal resolution. This is

generally better at bit rates of 192 and higher. Setting p to 0 always forces the ISO scale factor selection while setting p to .9999999 always forces the special joint stereo scale factor selection.

Parameter Q - joint stereo boundary adjustment

This parameter ranges from -7 to 7 and represents an adjustment to the sub-band where joint stereo starts. For example, if the psychoacoustic model chooses 14 for the start of the joint stereo and the Q parameter is set to -3, the joint boundary set to 11 (14 - 3). The joint bound must be 4, 8, 12 or 16 so the joint boundary is rounded to the closest value which is 12.

Parameter R - demand minimum factor

This value ranges from 0 to 1 and represents the minimum that the demand bit rate is allowed to be. For example, if the demand bit rate mode of bit allocation is used and the demand bit rate is set to a maximum of 256 kbs and the R parameter is set to .75 then the minimum bit rate is 192 kbs (256 * .75). This parameter should not be necessary if the model was completely accurate. When tuning with the demand bit rate, this parameter should be set to .25 so that the minimum bit rate is a very low value.

Parameter S - stereo used sub-bands

This parameter ranges from 0 to 31 where 0 means use the default maximum (27 or 30) sub-bands as specified in the ISO specification when operating in the stereo and dual mono modes. If this parameter is set to 15, then only sub-bands 0 to 14 are allocated bits and sub-bands 15 and above have no bits allocated. Setting this parameter changes the frequency response of the CODEC. For example, if the sampling rate is 48,000 samples per second, then the sub-bands represent 750 HZ of bandwidth. If the used sub-bands is set to 20, then the frequency response of the CODEC would be from 20 to 15000 HZ (20 * 750).

Parameter T - joint frame count

This parameter ranges from 0 to 24 and represents the minimum number of MUSICAM® frames (24 millisecond for 48k or 36 ms for 32k) that are coded using joint stereo. Setting this parameter non-zero keeps the model from switching quickly from joint stereo to dual mono. In the

ISO model, there are 4 joint stereo boundaries. These are at sub-band 4, 8, 12 and 16 (starting at 0). If the psychoacoustic model requires that the boundary for joint stereo be set at 4 for the current frame and the next frame can be coded as a dual mono frame, then the T parameter requires that the boundary be kept at 4 for the next T frames, then the joint boundary is set to 8 for the next T frames and so on. This prevents the model from switching out of joint stereo so quickly. If the current frame is coded as dual mono and the next frame requires joint stereo coding, then the next frame is immediately switched into joint stereo. The T parameter has no effect for entering joint stereo, it only controls the exit from joint stereo. This parameter attempts to reduce annoying artifacts which arise from the switching in and out of the joint stereo mode.

Parameter U - peak / rms selection

This is a binary parameter. If the value is less than .499, then the psychoacoustic model utilizes the peak value of the samples within each sub-band to determine the number of bits to allocate for that sub-band. If the parameter is greater than .499, then the RMS value of all the samples in the sub-band is used to determine how many bits are needed in each sub-band. Generally, utilizing the RMS value results in a lower demand bit rate and higher audio quality.

Parameter V - tonal masker addition

This parameter is a binary parameter. If it is below .499 the 3 db additional rule is used for tonals. If it is greater than .499, then the 6db rule for tonals is used. The addition rule specifies how to add masking level for two adjacent tonal maskers. There is some psychoacoustic evidence that the masking of two adjacent tonal maskers is greater (6db rule) than simply adding the sum of the power of each masking skirt (3db). In other words, the masking is not the sum of the powers of each of the maskers. The masking ability of two closely spaced tonal maskers is greater than the sum of the power of each of the individual maskers at the specified frequency. See FIGURE 6.

Parameter W - sub-band 3 adjustment

This parameter ranges from 0 to 15 db and represents an adjustment which is made to the psychoacoustic model for sub-band 3. It tells the psychoacoustic model to

allocate more bits than calculated for this sub-band. A value of 7 would mean that 7db more bits (remember that 1 bit equals 6 db) would be allocated to each sample in sub-band 3. This is used to compensate for inaccuracies in the psychoacoustic model at the frequency of sub-band 3 (3*750 to 4*750 Hz for 48k sampling).

Parameter X - adj sub-band 2 adjustment

This parameter is identical to parameter W with the exception that the reference to sub-band 3 in the above-description for parameter W is changed to sub-band 2 for parameter X.

Parameter Y - adj sub-band 1 adjustment

This parameter is identical to parameter W with the exception that the reference to sub-band 3 in the above-description for parameter W is changed to sub-band 1 for parameter Y.

Parameter Z - adj sub-band 0 adjustment

This parameter is identical to parameter W with the exception that the reference to sub-band 3 in the above-description for parameter W is changed to sub-band 0 for parameter Z.

Parameter KK - sb hang time

The psychoacoustic model may state that at the current time, a sub-band does not need any bits. The KK parameter controls this condition. If the parameter is set to 10, then if the model calculates that no bits are needed for a certain sub-band, 10 consecutive frames must occur with no request for bits in that sub-band before no bits are allocated to the sub-band. There are 32 counters, one for each sub-band. The KK parameter is the same for each sub-band. If a sub-band is turned off, and the next frame needs bits, the sub-band is immediately turned on. This parameter is used to prevent annoying switching on and off of sub-bands. Setting this parameter non-zero results in better sounding audio at higher bit rates but always requires more bits. Thus, at lower bit rates, the increased usage of bits may result in other artifacts.

Parameter LL - joint stereo scale factor adjustment

If this parameter is less than .49999, then scale factor adjustments are made. If this parameter is .5000 or greater, then no scale factor adjustments are made (this is the ISO mode). This parameter is used only if joint stereo is used. The scale factor adjustment considers the left and right scale factors a pair and tries to pick a scale factor pair so that the stereo image is better positioned in the left/right scale factor plane. The result of using scale factor adjustment is that the stereo image is significantly better in the joint stereo mode.

Parameter MM - mono used sub-bands

This parameter is identical to parameter S except it applies to mono audio frames.

Parameter NN - joint stereo used sub-bands

This parameter is identical to parameter S except it applies to joint stereo audio frames.

As the psycho-acoustic parameters affect the resultant quality of the audio output, it would be advantageous for users to vary the output according to the user's desires.

In a preferred embodiment of the disclosed CODEC 1010, the psycho-acoustic parameters can be adjusted by the user through a process called dynamic psycho-acoustic parameter adjustment (DPPA) or tuning. The software for executing DPPA is disclosed in the incorporated Software Appendix and discussed in more detail below in connection with Figs. 21-32. DPPA offers at least three important advantages to a user of the disclosed CODEC over prior art CODECs. First, DPPA provides definitions of the controllable parameters and their effect on the resulting coding and compression processes. Second, the user has control over the settings of the defined DPPA parameters in real time. Third, the user can hear the

result of experimental changes in the DPPA parameters. This feedback allows the user to intelligently choose between parameter alternatives.

Tuning the model parameters is best done when the demand bit rate is used. Demand bit rate is the bit rate calculated by the psycho-acoustic model. The demand bit rate is in contrast to a fixed bit rate. If a transmission facility is used to transmit compressed digital audio signals, then it will have a constant bit rate such as 64, 128, 192, 256 ... kbs. When tuning the parameters while using the Parameter N described above, it is important that the demand bit rate is observed and monitored. The model parameters should be adjusted for the best sound with the minimum demand bit rate. Once the parameters have been optimized in the demand bit rate mode, they can be confirmed by running in the constant bit rate mode (see Parameter N).

DPPA also provides a way for the user to evaluate the effect of parameter changes. This is most typically embodied in the ability for the user to hear the output of the coding technique as changes are made to the psycho-acoustic parameters. The user can adjust a parameter and then listen to the resulting change in the audio quality. An alternate embodiment may incorporate measurement equipment in the CODEC so that the user would have an objective measurement of the effect of parameter adjustment on the resulting audio. Other advantages of the disclosed invention with the DPPA are that the user is aware of what effect the individual parameters

have on the compression decompression scheme, is able to change the values of parameters, and is able to immediately assess the resulting effect of the current parameter set.

One advantage of the ability to change parameters in the disclosed CODEC, is that the changes can be accepted in real time. In other words, the user has the ability to change parameters while the audio is being processed by the system.

In the preferred embodiment, the compression scheme (attached as the Software Appendix) includes thirty adjustable parameters. It is contemplated that additional parameters can be added to the CODEC to modify the audio output. Provisions have been made in the CODEC for these additional parameters.

Turning now to FIGURE 19, one can see two tonal maskers 1024 and 1025. The individual masking skirts for these maskers are shown in 1028. The encoder predicts how do these individual maskers mask a signal in the region in between 1024 and 1025. The summing of the masking effects of each of the individual maskers may be varied between two methods of summing the effects of tonal maskers. These methods are controlled by Parameter V described above.

FIGURE 20 is illustrative of the steps the user must take to modify each parameter. As shown in FIGURE 20, the parameters are set to their default value (which may be obtained from one of several stored table) and remain at that value until the user adjusts the parameter. The user may change the parameter by

turning one of the knobs, pushing one key on the keypad, or changing one of the graphics representative of one of the parameters on the computer monitor. Thus, as shown in box 1060, the disclosed CODEC 1010 waits until the user enters a command directed to one of the parameters. The CODEC 1010 then determines which parameter had been adjusted. For example, in box 1062 the CODEC inquires whether the parameter that was modified was parameter J. If parameter J was not selected, the CODEC 1010 then returns to box f1060 and awaits another command from the user. If parameter J was selected, the CODEC 1010 awaits for the user to enter a value for that parameter in box 1064. Once the user has entered a value for that parameter, the CODEC 1010, in box 1066, stores that new value for parameter J. The values for the default parameters are stored on a storage medium in the encoder 1012, such as a ROM or other chip.

Turning again to FIGURES 14 and 15 (which generally illustrate the operation of the disclosed CODEC) an analog audio source 1016 is fed into the encoder/decoder (CODEC) 1010 which works in loop back mode (where the encoder directly feeds the decoder). Parametric adjustments can be made via a personal computer 1040 attached to the CODEC 1010 from an RS232 port (not shown) attached to the rear of the CODEC. A cable 1042 which plugs into the RS232 port, connects into a spare port (not shown) on the PC 1040 as shown in FIGURE 14. The personal computer 1040 is preferably an IBM-PC or IBM-PC clone, but can be an any personal computer

including a Mackintosh®. The personal computer 1040 should be at least a 386DX-33, but is preferably a 486. The PC should have a VGA monitor or the like. The preferred personal computer 1040 should have at least 4 mb of memory, a serial com port, a mouse, and a hard drive.

Once the PC 1040 is connected to the CODEC 1010, a tuning file can be loaded onto the personal computer 1040, and then the parameters can be sent to the encoder via a cable 1042. A speaker 1044 is preferably attached to the output of the CODEC 1010, via a cable 1046, to give the user real time output. As a result, the user can evaluate the results of the parameter adjustment. A headphone jack (not shown) is also preferably included so that a user can connect headphones to the CODEC and monitor the audio output.

The parameters can be adjusted and evaluated in a variety of different ways. In the preferred embodiment, a mouse is used to move a cursor to the parameter that the user wishes to adjust. The user then holds down the left mouse button and drags the fader button to the left or right to adjust the parameter while listening to the audio from the speaker 1044. For example, if the user were to move the fader button for parameter J to the extreme right, the resulting audio would be degraded. With this knowledge of the system, parameter J can be moved to test the system to insure that the tuning program is communicating with the encoder. Once the

user has changed all or some of the parameters, the newly adjusted parameters can be saved.

In another embodiment, control knobs or a keypad (not shown), can be located on the face of the CODEC 1010 to allow the user to adjust the parameters. The knobs would communicate with the tuning program to effectuate the same result as with the fader buttons on the computer monitor. The attachment of the knobs can be hard with one knob allotted to each adjustable parameter, or it could be soft with a single knob shared between multiple parameters.

In another embodiment, a graphic representing an "n" dimensional space with the dimensions determined by the parameters could be shown on the computer display. The operator would move a pointer in that space. This would enable several parameters to be adjusted simultaneously. In still another embodiment, the parameters can be adjusted in groups. Often psycho-acoustic parameters only make sense when modified in groups with certain parameters having fixed relationships with other parameters. These groups of parameters are referred to as smart groups. Smart group adjustment would mean that logic in the CODEC would change related parameters (in the same group) when the user changes a given parameter. This would represent an acceptable surface in the adjustable parameter space.

In yet another embodiment, a digital parameter read out may be provided. This would allow the values of the parameters to be digitally displayed on either the CODEC 1010 or the PC 1040. The

current state of the CODEC 1010 can then be represented as a simple vector of numbers. This would enable the communication of parameter settings to other users.

Parameter adjustment can be evaluated in ways other than by listening to the output of speaker 1044. In one embodiment, the CODEC 1010 is provided with an integrated FFT analyzer and display, such as shown in applicant's invention entitled "System For Compression And Decompression Of Audio Signals For Digital Transmission," and the Software Appendix that is attached thereto, that are both hereby incorporated by reference. By attaching the FFT to the output of the CODEC, the user is able to observe the effect of parametric changes on frequency response. By attaching the FFT to the input of the CODEC, the user is able to observe frequency response input. The user can thus compare the input frequency response to the output frequency response. In another embodiment, the disclosed CODEC 1010 is provided with test signals built into the system to illustrate the effect of different parameter adjustments.

In another embodiment, the DPPA system may be a "teaching unit." To determine the proper setting of each parameter, once the determination is made, then the teacher could be used to disburse the parameters to remote CODECs (receivers) connected to it. Using this embodiment, the data stream produced by the teaching unit is sent to the remote CODEC that would then use the data stream to synchronize their own parameters with those determined to be

appropriate to the teacher. This entire system thus tracks a single lead CODEC and avoids the necessity of adjusting the parameters of all other CODECs in the network of CODECs.

Processing Flow of the Preferred Embodiment

Next, the processing flow of the preferred embodiment is described in connection with Figs. 21-33.

Fig. 21 generally illustrates the functions of an encoder for a single channel receiving audio signal. The encoder includes a plurality of band pass filters separately divided into a low pass filter bank 502 and a high pass filter bank 504. The low and high pass filter banks 502 and 504 include a plurality of band pass filters 506. The number of band pass filters in each filter bank may be dynamically varied during joint stereo framing by the psycho-acoustic processor as explained below. For purposes of illustration, four filters have been dynamically assigned to the low pass filter bank 502, and the remaining filters have been assigned to the high pass filter bank 504. The band pass filters 506 receive a segment of predefined length (e.g., 24ms) of an incoming analog audio signal and pass corresponding subbands thereof. Each band pass filter 506 is assigned to a separate pass band having a unique center frequency and a corresponding bandwidth. The widths of each pass band may differ, for instance, whereby the band pass filters for low frequency signals have narrower pass bands than the pass bands of filters corresponding to

high frequency signals. The band pass filters are defined such that the pass bands slightly overlap.

The subband signals output by the band pass filters 506 are delivered to corresponding scalers 508 which adjust the gain of the subband signals and deliver same to corresponding quantizers 510. The subband signals received by each scaler 508 are divided into a predetermined number of blocks (e.g. three blocks each of which is 8 milliseconds in length for a 24 millisecond segment of audio data). The scalers 508 adjust the gain of the corresponding subband signal for each block within a segment until the peak to peak amplitude of the subband signal substantially corresponds to the range of the quantizer 510. The gain of the subband signal is controlled by the scaler 508 to ensure that the peak to peak amplitude never exceeds the capacity of the quantizer 510. By way of example, each subband signal delivered from a band pass filter 506 may include 36 samples divided into three blocks of 12 samples. The scaler 508 adjusts the gain of the 12 sample blocks as explained above to ensure that the quantizer 510 is fully loaded. The quantizer 510 has a maximum quantization capacity. The quantizers 510 convert the incoming samples to one of a predefined number of discrete levels and outputs a corresponding digital signal representative of the closest quantization level to the sample level. The number and distance between quantization levels is governed by the number of bits allocated to the quantizer 510. For instance, the quantizer 510 will use more quantization levels

if afforded 10 bits per sample as compared to the number of quantization levels which correspond to 6 bits per sample. As more bits are assigned to the quantizer, the sample is more accurately digitized and less noise is introduced. The quantizers 510 deliver output quantized subband signals to a multiplexer 512, which combines the subband signals to form a frame of data which is ultimately transmitted by the encoder.

A psycho-acoustic processor (PAP) 514 process the incoming analog audio signal (as explained below) and controls the quantizers 510 and scalers 508 to allocate the minimum necessary number of bits to each quantizer. In accordance with the process explained below, the PAP 514 may direct the quantizer 516 to utilize six bits per sample, while limiting quantizer 518 to two bits per sample.

Fig. 22 generally illustrates a frame 530 having a header segment 532, a data segment 534, and an ancillary data segment 536. The data segment 534 includes multiple subband components 538, each of which corresponds to a unique subband (SB_1-SB_{32}). Each subband component 538 is divided into three blocks 540, each of which has been scaled by the scaler 508 to properly load the quantizer 510. It is to be understood that the blocks 540 and subband components 538 will vary in length depending upon the number of bits used by the corresponding quantizer 510 to encode the corresponding subband signal. For instance, when quantizer 516 is directed (by the path 514) to use six bits per sample, the corresponding data component

542 will include 18 bits of data (six bits per block). However, when quantizer 518 is assigned two bits per sample, data component 544 will include six bits (two bits per block). The audio data segment 534 has a fixed maximum length, and thus a limited number of bits are available for use by the quantizers 510. The PAP 514 maximizes the bit allocation between the quantizers 510.

Once the bit allocation is complete, the PAP 514 loads the corresponding subsection and the header segment 532 with the corresponding encoder information 546. The encoder information 546 includes the number of bits allocated to each quantizer 510 for the corresponding subband (referred to hereafter as the "Bit Allocation Information 548). The encoder information 546 furth r includes the scaling factors 550 used by the scalers 508 in connection with corresponding blocks 540 of corresponding subband components 538. In addition, the encoder information 546 includes scaling factor sample information 552 (explained below).

Fig. 23 illustrates an encoder including the structure of the encoder from Fig. 21, with the further ability to offer joint stereo at a decoder output. In Fig. 23, the encoder is generally denoted by block 600, and the decoder is denoted by block 602. The encoder 600 receives a stereo signal upon left and right channels. The decoder 602 outputs a joint stereo signal at speakers 604 and 606. The encoder 600 includes low pass filter banks (LPFB) 608 and 612 corresponding to the left and right channels, respectively. The encoder 600 further includes high pass filter banks (HPFB) 610

and 614, also corresponding to the left and right channels, respectively. The low and high pass filter banks 608-614 include a plurality of band pass filters which are controlled by a PAP, as explained in connection with Fig. 21. The output signals of the low pass filter banks 608 and 612 are delivered to scaler banks 616 and 618, each of which also include a plurality of scalers which operate in a manner similar to the scalers 508 in Fig. 21. The scaler banks 616 and 618 deliver scaled signals to quantizer banks 620 and 622, each of which similarly includes a plurality of quantizers similar to quantizers 510 in Fig. 21.

While not showing, it is understood that the filter banks 616 and 618 and the quantizers 620 and 622 controlled by a PAP similar to the psycho-acoustic processor 514 in Fig. 21. The low pass filter banks 608 and 612, scaler banks 616 and 618, and quantizer banks 620 and 622 cooperate to separately encode the lower subbands for the left and right channels of the stereo input signal. The encoded signals for the lower subbands are in turn delivered from the quantizers 620 and 622 and ultimately received by corresponding inverting quantizers 624 and 626. The inverting quantizers 624 and 626 cooperate with inverse scaling banks 628 and 630 to reconvert the lower frequency portions of the encoded left and right channel signals back to analog audio.

The encoder 600 further includes a summer 632 which combines the output signals from the high pass filter banks 610 and 614 for the left and right channels to produce a joint mono signal for the

higher pass bands. The output of the summer 632 is in turn delivered to a scaling bank 634, which scales the signal to properly load the quantizer bank 636. The output signal of the quantizer bank 636 is delivered to an inverse quantizer 638 to reverse the process. The output of the inverse quantizer 638 is delivered to two scaling banks 640 and 642 which are controlled via control channels 644 and 646.

The encoder 600 further includes calculating modules 650 and 652, which measure the energy in the corresponding high pass subbands. The modules 650 and 652 then adjust the gain of scalers 640 and 642 in proportion to the energy of the corresponding high pass subbands. For instance, if HPFB 610 outputs more energy than HPFB 614, then scaler 640 is set to boost the gain of its input signal greater than the gain boost of scaler 642. Thus, the audio signal in the higher pass bands is output predominantly at speaker 604. The energy calculator 650 and 652 may be carried out by the psycho-acoustic processor in a manner explained below.

Next, the discussion turns to the process followed by the present invention to undergo encoding.

With reference to Fig. 24, the PAP 514 cooperates with the quantizer 510 and scaler 508 to digitize the analog audio signals received from each band pass filter 506 for corresponding subbands (step 2400). In step 2402, the digitized signals for the subbands from each bandpass filter are divided into a predefined number of blocks. For example, a 24 millisecond segment of analog audio may

be converted to 36 digital samples and then divided into three blocks of 12 samples each. In step 2404, each block of samples is analyzed to determine the maximum amplitude of the digitized signal therein. In step 2406, the scalers 508 are adjusted to vary the scale of the samples within each block until the samples correspond to a signal gain substantially equalling the range of the quantizers 510.

Turning to Figs. 25A and 25B, while the scalers 508 are being adjusted (as explained in connection with Fig. 24), the PAP 514 calculates the global masking threshold (GMT) to be used in connection with the present sample of analog audio data. Beginning at step 2502, the PAP 514 obtains a working table of psycho-acoustic parameters having a value for each of parameters A-NN (described above). The table of parameters may be one of several predefined tables stored in memory in the encoder. The table is updated dynamically by the user during operation of the encoder. For instance, when the encoder is initially started, an initial set of parameter values may be read from the encoder memory and used to initialize the encoder. Thereafter, as the PAP 514 continuously processes segments of analog audio data, the user may vary the parameter values stored in the working table. Once the user varies a parameter value in the working table, the PAP 514 obtains the new parameter value set for processing the following analog audio segments. For instance, after the user listens to a short segment (one minute) of analog audio encoded and decoded according to the

initial working table, the user may desire to adjust the parameters within the working table. Once the user adjusts these parameters, the PAP 514 effects subsequent psycho-acoustic processing based on the new parameter values assigned by the user. Thus, the user is afforded the opportunity to listen to the signal which results from the users adjustment in the parameters.

Returning to Fig. 25A, once the PAP 514 obtains the working table of parameters A-NN, the PAP 514 uses these parameter values for the current segment of audio data. At step 2504, the PAP 514 obtains a segment of analog audio data of predetermined length (e.g., 24 milliseconds). The segment is digitized. At step 2506, the PAP 514 converts the digitized segment from the time to the frequency domain according to the bark scale. These conversions may be effected using a Fast Fourier Transform and a known Bark transfer function between the bark frequency domain and the normal frequency domain. At step 2508, the PAP calculates the threshold of hearing. At step 2510, the PAP analyzes the signal converted in step 2506 to the bark frequency domain to locate the tonal peaks therein. Once located, the tonal peaks are removed in step 2512 from the digital converted signal. Next, the digitized signal is divided into critical bands (step 2514). Noise maskers are calculated for each critical band by summing the remaining energy within each critical band (after the tonal peaks have been removed). A representative noise masker is obtained for each critical band from the noise calculated in step 2514. It is

understood that, a signal noise masker is substituted therefore at a single frequency and having a predetermined amplitude. The amplitude and frequency of the noise masker are determined by the amount of noise energy within the critical band.

At step 2516 (Fig. 25B), the PAP calculates masking skirts for the tonal and noise maskers based on parameters A-J and based on the amplitudes and frequencies of the tonal and noise maskers. At step 2518, the PAP combines the noise and tonal masking skirts and the threshold of hearing to obtain a global masking threshold for the presently digitized segment of audio data. The global masking threshold (GMT) is divided into subbands at step 2520. The subbands correspond to the band pass filters 506. At step 2520 the PAP locates the maximum and minimum of each global masking threshold within each subband. At step 2522 the PAP assigns quantization levels for each subband based on amount of noise which may be added to each subband without exceeding the minimum value of the GMT within the corresponding subband. The assignment process is described in more detail below.

Turning to Fig. 26, the process of obtaining the GMT is explained in more detail. At step 2600, the PAP locates the first subband (subband 0) and obtains the first masker within this subband (step 2602). At step 2604, the PAP combines the current masker obtained in step 2602 with the threshold of hearing to obtain an initial GMT for the subband. Thereafter the next masker is obtained at step 2606. The PAP then determines at step 2608

whether the newly obtained and preceding maskers represent adjacent tonal maskers. If two adjacent tonal maskers are being combined, control flows to step 2610 at which the PAP combines the two adjacent total maskers within the GMT using one of two addition rules defined by parameter V. For instance, the two tonal maskers may be combined according to a 3db or a 6db addition rule based upon which is chosen by the parameter V. The tonal maskers are combined according to one of the following equations:

$$3db(rule) = 10\ log12(10\ P_{1(db)}\backslash 10 + 10P_{2(db)}\backslash 10)$$

$$6db(rule) = 2\ log12(1\ P_{1(db)}\backslash 2 + 1P_{2(db)}\backslash 2)$$

Returning to step 2608 if the two maskers are not tonal maskers, flow moves to step 2612 at which the maskers are combined with the global masking threshold according to the conventional method. Next, at step 2614 it is determined whether the current masker represents the last masker in the subband. If not, steps 2606-2612 are repeated. If the current masker represents the last masker in the subband, flow passes to step 2616 at which the PAP determines whether the current subband is one of subbands 0, 1, 2 and 3. If so, control passes to step 2618 at which the global masking threshold for the current subband is adjusted by a biasing level determined by the corresponding one of parameter W-Z. For instance, if the current subband is subband 2, then the GMT within subband 2 is adjusted by a db level determined by parameter Y. At step 2620 it is determined whether the last subband has been analyzed. If not, flow pass to step 2602 where the above described

processes repeated. Otherwise, control returns to the main routine illustrated in Fig. 25.

Next, the quantization level assignment process of step 2522 is described in more detail in connection with Fig. 30. The assignment process involves three primary phases, namely an allocation phase, a deallocation phase and an excess bit allocation phase. During the allocation phase step (3000), the PAP steps through each subband for each channel (left and right) and assigns the corresponding quantizer a number of bits to be used for quantizing the subband signal. During bit allocation, the number of bits allocated to a subband are incremented in predefined allocation steps until a sufficient number of bits are assigned to the quantizer to ensure that the noise introduced into the signal during the quantizing process is below the minimum of the GMT for the subband. Once the necessary number of bits are assigned to each subband at step 3000 it is determined whether the number of bits allocated has exceeded the number of bits available (i.e., the bit pool) at step 3002. If not, and extra bits exist then control flows to step 3004. At step 3004, the PAP determines whether the encoder is operating in a demand or constant bit rate mode. In a demand mode, once the PAP allocates bits to each subband, the allocations become final, even through the total number of bits needed is less than the number available for the current transmission rate. Thus, the allocation routine ends. However,

when in a constant bit rate mode, the extra bits are distributed evenly or unevenly among the subbands.

It is desirable to choose the demand bit rate made when tuning the codec to ensure that the signal heard by the user accurately reflects the parameter values set by the user. The remaining bits from the bit pool are distributed amongst the subbands to further reduce the quantization noise. However, if bit allocation in step 3000 has exceeded the bit pool then flow passes to step 3006 at which bit deallocation is performed and previously assigned bits are removed from selected quantizers which are deemed the best candidate for deallocation. Deallocation occurs with respect to those subbands at which deallocation will have the least negative effect. Put another way, the PAP deallocates bits from subbands which will continue, even after deallocation, to have quantization noise levels closest to the GMT minimum for that subband (even though the quantization noise level exceeds the GMT minimum).

During bit allocation, flow passes at step 3000 to the routine illustrated in Fig. 27. At step 2702, the PAP determines whether the encoder is operating in a stereo, mono, or joint stereo framing mode. The PAP sets the last subband to be used which is determined by the subband limit parameters S, MN and NN. At step 2704, the PAP determines the total number of bits available (i.e., the bit pool) for the current framing mode, namely for joint stereo, stereo or mono. At step 2706, the first subband and first channel are obtained. At step 2708, the maximum for the signal within the

current subband is compared to the GMT minimum within the current subband. If the subband signal maximum is less than the GMT minimum, then the current subband signal need not necessarily be transmitted since it falls below the GMT. Thus, flow passes to step 2710 at which it is determined whether the current subband falls below a subband limit (defined by parameter M). If the current subband is below the subband limit then the PAP allocates bits to the subband even through the subband signal falls below the GMT minimum. For instance, if the current subband is two and the user has designated (via parameter M) that subbands 0-5 should be encoded and transmitted, then subband 2 would be encoded by the corresponding quantizer with a minimum number of bits allocated to the quantizer. Thus, at step 2710, if the current subband is less than the subband limit then control passes to step 2712 at which the bit allocation routine is called to assign at least a first allocation step of a minimum number of bits to the current subband. However, at step 2710 if it is determined that the current subband is greater than the subband limit then control passes to step 2718 and the bit allocation routine is bypassed (i.e. the quantizer for the current subband is not assigned any bits and thus the signal within the current subband is not encoded, nor transmitted). At step 2712, prior to performing the bit allocation routine, the digitized audio signal within the current subband is adjusted to introduce a safety margin or bias thereto to shift the digitized

signal upward or downward. This safety margin represents a parameter adjusted dynamically by the user (parameter O).

After flow returns from the bit allocation routine, it is determined at step 2714 whether the encoder is operating in a joint stereo mode. If not flow passes to step 2718 at which it is determined whether the foregoing process (steps 2708-2714) need to be repeated for the opposite channel. If so, the channels are switched at step 2724 and the process is repeated. If not, flow passes from step 2718 to 2722 at which it is determined whether the current subband is the last subband. If not, the current subband is incremented at step 2726 and the allocation routine is repeated. Thus, steps 2708-2726 are repeated each subband.

Returning to step 2714, when operating in a joint stereo mode, control passes to step 2716 at which it is determined whether the bit allocation routine at step 2712 allocated a number of bits to the current subband which resulted in the total number of allocated bits exceeding the available bit pool for the current mode. If so, the current subband number is recorded at step 2720 as the subband at which the bit pool boundary was exceeded.

When in a stereo mode the process flows from step 2708 to step 2726 without using steps 2716 and 2720 in order that every subband within the right and left channels is assigned the necessary number of bits to insure that the quantization noise falls below the global masking threshold within the corresponding subband. When in the joint stereo mode, the foregoing process is repeated separately

for every subband within the left and right channels (just as in the stereo mode). However, the system records the subband number at which the available bit pool was exceeded in step 2720. This subband number is later used to determine a joint stereo boundary such that all subbands below the boundary are processed separately in stereo for the left and right channels. All subbands above the boundary are processed jointly, such as shown by the joint stereo encoder of Fig. 23. The subband boundary corresponds to the break point between the low pass filter banks 608 and 612 and the high pass filter banks 610 and 614 (shown in Fig. 23).

Turning to Fig. 28, the bit allocation routine is described in more detail. Beginning at step 2802, an array of allocation steps is obtained for the current mode (e.g., stereo, mono or joint stereo. Each level within the array corresponds to a predefined number of bits to be assigned to a quantizer. By way of example, the array may include 17 elements, with elements 1, 2 and 3 equaling 60 bits, 84 bits and 124 bits, respectively. Thus, at the first step 60 bits are assigned to the quantizer corresponding to the current subband. At the second step, 84 bits are assigned to the quantizer corresponding to the current subband. Similarly, at the third step, 124 bits are assigned to the quantizer for the current subband. The steps are incremented until the current step allocates a sufficient number of bits to the quantizer to reduce the quantization noise below the minimum GMT for the current subband. In addition to the bit allocation array, a mask to noise

ratio array is included containing a list of elements, each of which corresponds to a unique step. Each element contains a predefined mask to noise ratio identifying the amount of noise introduced into the encoded signal when a given number of bits are utilized to quantize the subband. For instance, steps 1, 2 and 3 may correspond to mask to noise ratios (MNR) of 10db, 8db and 6db, respectively. Thus, if 60 bits are allocated to the current quantizer for quantizing the current subband, 10db of noise will be introduced into the resultant encoded signals. Similarly, if 84 bits are used to quantize the signal within the current subband, 8db of noise are introduced.

At step 2802, the allocation and MNR arrays are obtained and the current step is set to 1. At step 2804, the allocation array is accessed to obtain the number of bits to be allocated to the current subband for the current step. At step 2806 the maximum level of the audio signal within the current subband is obtained based on one of the audio peak or RMS value, which one selected between determined by parameter U. Next, the MNR value for the current step is obtained from the MNR array (2808). At step 2810, it is determined whether the audio signal maximum, when combined with the MNR value of the current allocation step, exceed the minimum of the GMT for the current subband. If so, then a detectable amount of noise will be introduced into the signal if the current allocation step is used. Thus, control passes to step 2816.

At step 2816, the PAP records the difference between the GMT minimum of the current subband and the level combined signal formed from the maximum value for the audio signal and the MNR. Thereafter, at 2818 the allocation step is incremented in order to allocation more bits to the current subband. The foregoing loop is repeated until the allocation step is incremented sufficiently to allocate a number of bits to the current subband necessary to reduce the combined signal formed from the audio signal max and MNR below the minimum of the GMT. Once it is determined at step 2810 that this combined signal is less than the minimum of the GMT, control passes to step 2812. At step 2812, the number of bits corresponding to the current step are allocated to the quantizer for the current subband. At step 2814, the system updates the total number of allocated bits for the current segment of audio information.

According to foregoing process, each quantizer is assigned a number of bits corresponding to an allocation step which is just sufficient to reduce the combined noise and audio signal below the minimum of the GMT. In addition, at step 2816, the system retains a deallocation table having one element for each subband and channel. Each element within the table corresponds to the difference between the GMT minimum and the combined audio signal maximum and MNR value for the allocation step preceding the allocation step ultimately assigned to the quantizer in step 2812.

By way of example, a quantizer may be assigned the number of bits corresponding to allocation step 3 (e.g., 124 bits). At step 2816, it was determined that the signal and MNR for step 2 exceeded the GMT minimum by 3db. The deallocation table will record at step 2816 this 3db value indicating that, while the current quantizer is assigned to allocation step 3, if the current quantizer had been assigned to allocation step #2, the combined signal and MNR would exceed the GMT minimum by 3db. The deallocation table recorded at step 2816 may be used later if the deallocation of bits becomes necessary (as explained below).

The bit allocation routine of Fig. 28 is continuously repeated for each channel and for each subband (according to the process of Fig. 27). Once control returns to step 3000 in Fig. 30, all of the subbands for both channels have been allocated the necessary number of bits. At step 3002 if it is determined that the number of bits allocated exceeds the bit pool, control passes to step 3006 which is illustrated in more detail in Fig. 31.

When it is determined that deallocation is necessary, control passes from step 3006 (Fig. 30) to the deallocation routine illustrated in Fig. 31. At step 3102, it is determined whether the encoder is operating in a joint stereo mode. If so, control passes to step 3104 at which the joint stereo boundary is determined. The joint stereo boundary represents the boundary between the low pass filter banks 608 and 612 and high pass filter banks 610 and 614 (Fig. 23). Subbands below the joint stereo boundary are processed

separately for the left and right channels within the low pass filter banks 608 and 612. Subbands above the joint stereo boundary are included within the high pass filter banks 610 and 614 and are combined in summer 632 to form a mono signal. Thus, subbands above the joint stereo boundary are combined for the left and right channels and passed through a single quantizer bank 636.

Returning to Fig. 31, once the joint stereo boundary is determined, a new bit pool is obtained based on the joint stereo boundary (step 3106). A new bit pool must be calculated since the original bit pool which calculated based on full stereo whereby it was presumed that bits would be allocated to all of the subbands separately for the left and right channels. However, subbands above the boundary are combined for the left and right channels and thus additional bits are available for allocation. For instance, in a full stereo system using 22 subbands per channel, bits must be allocated between 44 separate subbands (i.e., 22 subbands for the left channel and 22 subbands for the right channel). However, in a joint stereo mode utilizing 22 subbands with the joint stereo boundary at subband 8, only 32 subbands are necessary (i.e., eight lower subbands for the left channel, eight lower subbands for the right channel and 16 upper subbands for the combined signals from the left and right signals). Once the new bit pool is calculated, the joint stereo array is obtained at step 3108. The joint stereo array identifies the allocation steps combining the number of bits to be allocated for each step during the bit allocation routine

(Fig. 28). In addition, the joint stereo array identifies the mask to noise ratio for each allocation step. At step 3110, the bit allocation routine (Fig. 28) is called to allocate bits to the subbands, wherein subbands below the joint stereo boundary are separately allocated for the left and right channels, while subbands above the joint stereo boundary are allocated for a single set of band pass filters representing the combination of the signals from the left and right channels.

Next, at step 3112, it is determined whether the bit allocation for the joint stereo frame exceeds the joint stereo bit pool (obtained at step 3106). If not, control returns to the routine in Fig. 30. Howev:r, if more bits have been allocated than are available in the bit pool, control passes to step 3114 to begin a deallocation process. At step 3114, the deallocation table (generated at step 2816 in Fig. 28) is sorted based on the difference values recorded therein to align these difference values in descending order. At step 3116, the first element within the deallocation table is obtained. At step 3118, a deallocation operation is effected. To deallocate bits, the quantizer corresponding to the channel and subband identified in the first element of the deallocation table is assigned a new number of quantizing bits. The number of bits newly assigned to this quantizer corresponds to the step preceding the step original assigned to the quantizer. For instance, if during the original allocation routine, a quantizer was assigned 124 bits

(corresponding to step 3), then at step 3118, the quantizer would be assigned 84 bits (corresponding to allocation step 2).

At step 3120, a new difference value is calculated for the current subband based on the allocation step preceding the newly assigned allocation step. This new difference is added to the difference table at step 3122. The number of deallocated are then subtracted from the allocated bit total (step 3124). Thereafter, it is determined whether the new total of bits allocated still exceeds the available bit pool (step 3126). If not, control returns to step 3006 (Fig. 30). If the allocation bit total still exceeds the bit pool, control returns to step 3114 and the above described deallocation processes is repeated.

Figs. 32 and 33 set forth an example explained hereafter in connection with the allocation steps and deallocation routine. Figs. 32A and 32B illustrate two exemplary subbands with the corresponding portions of the global masking threshold and the quantized signal levels derived from the audio signal peak and MNR value. The quantized signal levels are denoted at points 3106-3108 and 3110-3113. The minimums of the GMT are denoted at levels 3204 and 3205. Stated another way, if the number of bits associated with allocation step #1 are assigned to the quantizer for subband 3 (Fig. 32A), the resultant combined audio signal and MNR will have a magnitude proximate line 3206. If more bits are assigned to the quantizer (i.e., allocation step #2), the combined signal and MNR value is reduced to the level denoted at line 3207. Similarly, at

allocation step #3, if additional bits are allocated to the quantizer the combined audio signal and MNR value will lie proximate line 3208.

With reference to Fig. 32B, at allocation step #1 the combined audio and MNR level will lie proximate line 3210. At step #2, the it will be reduced to level 3211, and at allocation step #3, it will fall to line 3212. At allocation step 4, sufficient bits will be allocated to the quantizer to reduce the combined signal and MNR value to level 3213 which falls below the GMT min at point 3205.

The bit allocation routine as discussed above, progresses through the allocation steps until the combined signal and MNR value (hereafter the quantizing valve) falls below the minimum of the GMT. During each innervation through the bit allocation routine, when the quantizing value is greater than the GMT min, the deallocation table is updated to include the difference value between the minimum of the GMT and the MNR value. Thus, the deallocation table of Fig. 32 stores the channel and subband for each difference value. In the present example, the deallocation table records for subband 3 (Fig. 32A) the difference value 3db which represents the distance between the minimum of the GMT at point 3204 and the quantization level at point 3207 above the GMT. The table also stores the allocation step associated with the quantization value at line 3207. The deallocation table also stores an element for subband 7 which represents the difference

value between the minimum of the GMT and the quantization level corresponding to line 3212.

During the deallocation routine, the deallocation table is resorted to place with the difference values in ascending order, such that the first element in the table corresponds to the subband with the least difference value between the minimum GMT and quantization level of the next closest MNR value. The quantizer corresponding to subband 7 is deallocated, such that the number of bits assign thereto is reduced from the number of bits corresponding to step #4 (line 3213) to the number of bits corresponding to step #3 (line 3212). Thus, the deallocation routine subtracts bits from the subband which will introduce the least amount of noise above the GMT for that subband. Once the subband 7 has been deallocated, the difference value is recalculated for the next preceding step (corresponding to MNR at line 3211). This new difference value is stored in the deallocation table along with its corresponding allocation step. If the number of bits deallocated during the first pass through this process is insufficient to lower the total allocated bits below the available bit pool maximum, than the processes repeated. In a second innervation, the quantizer corresponding to subband 3 would be reallocated with fewer bits corresponding to allocation step #2 (line 3207). This process is repeated until the total allocated bits falls within the available bit pool.

Basic Components and CODEC System

Figure 1 illustrates a high level block diagram of a CODEC 1. Figure 1 shows an encoder digital signal processor (DSP) 1, a decoder DSP 2, an LED DSP 95, an asynchronous multiplexer 3, an asynchronous demultiplexer 6, at least one digital interface module (DIM) 7 connected to the encoder output, at least one DIM 8 connected to the decoder input, a loopback control module 9, and a control processor 5. The encoder 1 inputs digital signals and timing signals and outputs compressed audio bit streams. The decoder 2 similarly inputs compressed audio bit streams and timing signals and outputs decompressed digital signals.

The CODEC 1 is capable of holding several audio compression algorithms (e.g. ISO MPEG and G.722). These and other algorithms might be downloaded into the CODEC from ISDN and thus future upgrades are simple and effortless to install. This creates an extremely versatile CODEC that is resistant to obsolescence. This should be contrasted to the ROM type of upgrade procedure currently employed by most CODEC manufacturers.

The CODEC 1 may also use a unique compression technique which is explained below and is described in the attached Software Appendix. This compression technique also uses an increased number of psycho-acoustic parameters to facilitate even more efficient compression and decompression of audio bit streams. These additional parameters are described above.

The CODEC 1 also contains a control processor 5 for receiving and processing control commands. These commands are conveyed to the various CODEC 1 components by a line 51. These commands might be entered by a user via front panel key pads such as 15, 152, and 154, as shown in Figures 5, 6, and 7. Keypad commands enter processor 5 through a line 52. The keypad also allows the user to navigate through a menu tree of command choices which fall into the general categories of common commands, encoder commands, decoder commands, and maintenance commands. Such menu choices are displayed on a Front Panel LCD display (not shown) via signals from a processor 5 on a line 58. (See LCD Menu Summary of commands, Chap 8 of CODEC manual, attached to the end of this specification before the claims). The LCD display might also be used for characters to show responses to front panel user commands as well as spontaneous messages such as incoming call connect directives. Additionally, the LCD display may be used to display graphical information.

The CODEC processor 5 may receive commands from a front panel remote control panel (RS232 interface format) and enter the processor 5 through the line 54. A front panel remote control allows computer access to all internal functions of the CODEC 1. Front panel remote control is especially useful for applications that need quick access via a palm top or lap top computer. This frequently occurs in control rooms where there are many CODECs in equipment racks serving different functions. A full complement of

remote control commands exists to facilitate control of the CODEC 1 (See the listing of remote control commands from the "cdqPRIMA" operating manual, Chapter 9, attached to the end of specification).

Referring again to Figure 2, this more detailed block diagram of CODEC 1 shows external front panel remote control data interacting with Front Panel Remote Control UART 178 via a line 54. UART 178 is controlled by the Control Micro 5 via a control network line 155.

The CODEC 1 also provides a rear panel remote control port which uses either RS232 or RS485 interface formats. The RS485 port may be either a 2 or 4 wire interface. A rear panel remote control also allows computer access to all the internal functions of the CODEC 1. Rear panel remote control is especially useful for applications which need permanent access to the CODEC 1 via computer control. This frequently occurs when the CODEC 1 is remotely located from the control room. The electrical interface choice is controlled by a command entered through remote control or a keypad.

Referring again to Figure 2, this more detailed block diagram of the CODEC 1 shows external rear panel remote control data interacting with Remote Control UART 18 via line 56. UART 18 is controlled by Control Micro 5 via the control network line 155.

The CODEC also includes a Front Panel LED display 3, examples of which are shown in Figures 11 and 12. This includes a set of

Status, Encoder, and Decoder LED's to show the status of various CODEC functions, for instance which compression algorithm is being used, and/or whether error conditions exist. The Status 31, Encoder 32, and Decoder 33 groups of LED's might be independently dimmed to allow emphasis of a particular group.

Referring again to Figure 1, signals from control processor 5 enter LED DSP 95 through the line 51. These control signals are processed by a LED DSP 95 and drive a LED display 3 (Figures 11 and 12) via a line 96.

A LED display 3 also shows peak and average level indications for the encoder 32 (left and right channels) and the decoder 34 (left and right channels). Each LED represents 2 dB of signal level and the maximum level is labeled dB. This maximum level is the highest level permissible at the input or at the output of the CODEC. All levels are measured relative to this maximum level. The level LED's display a 4 dB audio range. A peak hold feature of the level LED's shows the highest level of any audio sample. This value is instantly registered and the single peak level LED moves to the value representing this signal. If the peak level of all future signals are smaller, then the peak LED slowly decays to the new peak level. The peak level LED utilizes a fast attack and slow decay operation. The LED display 3 also includes a level display to show stereo image 36 which is used to display the position of the stereo image. This is useful when setting the levels of the left and right channels to insure the proper balance. Also

included is a correlation level display 38 which is used to check if the left and right channels are correlated. If the left and right channels are correlated, then they might be mixed to mono. The level LED's might also be used to display a scrolling message.

Referring again to Figure 2, this more detailed block diagram of CODEC 1 shows the LED DSP 95 driving a LED Array 125 via a connection 96. As also shown, the LED DSP 95 is controlled by the Control Micro 5 via the control network line 155. The DSP 95 also drives an Headphone (Hp) D/A Converter 98 via a connection 97. A converter 98 then outputs this analog signal via a connector 99 to external headphones (not shown). The headphones allow the user to monitor both the input and output signals of the CODEC 1. Figures 11 and 12 show headphone indicators 31 at the far right of the level displays to denote the signal output to the headphones. If both LED's are illuminated, then the left audio channel is output to the left earphone and the right audio channel is output to the right earphone. If only the left LED is illuminated, the left audio channel is output to both the left and right headphone. Similarly, if only the right LED is illuminated, the right audio channel is output to both the left and right headphone.

Analog Inputs and Outputs

Figure 2 shows a more detailed block diagram of the CODEC 1 structure. Referring to Figures 1 and 2, the left audio signal 12

and the right audio signal 14 are external analog inputs which are fed into an Analog to Digital (A/D) Converter 1, and converted into digital signals on a line 11. Similarly digital audio output signals on a line 121 are converted from Digital to Analog (D/A) via a converter 15. The converters 1 and 15 use an 18 bit format. The analog sections of the CODEC are set to +18 dBu maximum input levels. Other analog input and output levels might used.

Direct Digital Inputs and Outputs

Referring again to Figure 1, the CODEC 1 also allows for direct input of digital audio information via an AES/EBU digital audio interface on line 16 into encoder 1. The decoder 2 similarly outputs decoded, decompressed digital audio information on AES/EBU output line 22. These interfaces allow for interconnection of equipment without the need for A/D conversions. It is always desirable to reduce the number of A/D conversions since each time this conversion is performed, noise is generated. These interfaces might use a DB9 or XLR connectors.

AES/EBU digital input and output rates might vary and therefore such rates are converted, or adapted, by a Sample Rate Converter 11, to eliminate any digital clock problems. The A/D Converter 1 signals are similarly converted, or adapted, by a Sample Rate Convertor 11 before entering the encoder 1. Because of the rate adapters, the input/output digital rates are not required to be the same as the internal rates. For example, it is possible

to input 44.1 kHz AES/EBU digital audio input and ask the CODEC 1 to perform compression at 48, 44.1 or 32 kHz (by using the front panel LCD display or a remote control command). This is possible because of the digital rate adapters. Similarly, digital audio input sources might be 32, 44.1, or 48 kHz. These input sampling rates are automatically sensed and rate adapted. The compression technique at the encoder determines the internal digital sampling rate at the decoder, and a control command is used to set this rate. The AES/EBU digital output sampling rate from the decoder is also set via a control command and might be a variety of values.

The digital audio is output from the decoder at the sampling rate specified in the header. This rate might then be converted to other rates via the Sample Rate Convertor 12. The Sample Rate Convertors 11, 12 are capable of sampling rate changes between .51 and 1.99. For example, if the receiver received a bit stream that indicated that the sampling rate was 24 kHz, then the output sampling rate could be set to 32 or 44 kHz but not 48 kHz since 48 kHz would be a sampling rate conversion of 2. to 1. This is out of the range of the sampling rate converter. The allowed output sampling rates include 29.5, 32, 44.1, and 48 kHz. Other direct digital I/O formats might include, for example, SPDIF or Optical.

The encoder 1 receives direct digital input via a connector on the rear panel (line 16). Analog or digital signals (but not both simultaneously) may be input into the CODEC 1 as selected by a front panel switch. If the digital input is selected, the CODEC 1

locks to the incoming AES/EBU input and displays the lock condition via a front panel LED. If digital audio input is selected, an AES phase-lock loop (PLL) is used to lock onto the signal. Accordingly, the AES PLL lock light must be illuminated before audio is accepted for encoding. In normal operation, the CODEC 1 locks its internal clocks to the clock of the telephone network. For loopback (discussed below), the CODEC 1 locks its clocks to an internal clock. In either case, the clock used by the CODEC 1 is not precisely the same frequency as the AES/EBU input. To prevent slips from occurring due to the presence of two master clocks, a rate synchronizer is built into the encoder section to perform the necessary rate conversion between the two clocks.

The decoder 2 outputs direct digital signals via a rear panel connector (line 22). Additionally, the decoder may be synchronized to an external clock by an additional connector (SYNC, line 24) on the rear panel. Referring also to Figure 8, a block diagram is shown of the decoder output timing with the AES/EBU SYNC (line 24). disabled or not present during normal timing. If no input is present on the decoder AES/EBU SYNC input line 24 (Figure 1), then the output AES/EBU digital audio is generated by the internal clock source 2 that is either at the telephone or internal clock rate. Figure 9 additionally shows a block diagram of the decoder output timing with the AES/EBU SYNC disabled or not present, and using internal crystal timing.

Referring to Figure 1, a block diagram is shown of the decoder output timing with the AES/EBU SYNC (line 24) enabled and present using AES timing. If the SYNC input is present, then the digital audio output is generated at the frequency of the SYNC input via the clock generator 25 being fed into the rate adaptor 252. This adapted rate is used by the D/A Converter 254, as well as the AES/EBU transmitter and receiver units 256, 258. The presence of a valid sync source is indicated by illumination of the front panel AES PLL LED. The sync frequency many be slightly different from that of the CODEC 1 clock source and again the rate synchronism is performed to prevent any undesired slips in the digital audio output. The SYNC input is assumed to be an AES/EBU signal with or without data present. The CODEC 1 only uses framing for frequency and sync determination.

Referring again to Figure 2, this more detailed block diagram of CODEC 1 shows external digital input 16 entering AES/EBU receiver 13. The receiver output 14 then enters the Sample Rate Converter 11 and the rate is converted, if necessary, as described above. The converter 11 then feeds the rate adjusted bit stream via a line 111 into the encoder 1 for coding and compression.

Conversely, Figure 2 also shows the Decoder DSP 2 outputting a decoded and decompressed bit stream via a line 123 into the Sample Rate Converter 12. The converter 12 adapts the rate, if necessary, as described above and outputs the rate adjusted bit stream via line 122 into a AES/EBU Transmitter 126. The

transmitter 126 then outputs the digital signal through an external connection 22.

Figure 2 also shows the AES/EBU digital synchronous input line 24 leading into a AES/EBU Receiver 146. The receiver 146 routes the received SYNC input data into the Sample Rate Converter 12 via a line 147. The converter 12 uses this SYNC input for rate adapting as described above.

Asynchronous Ancillary Data

The CODEC 1 is also capable of handling a variety of ancillary data in addition to primary audio data. The audio packet, for instance, consists of a header, audio data, and ancillary data. If the sampling rate is 48 KHz, then the length of each packet is 24 milliseconds. The header consists of a 12 bit framing pattern, followed by various bits which indicate, among other things, the data rate, sampling rate, and emphasis. These header bits are protected by an optional 16 bit CRC. The header is followed by audio data which describes the compressed audio signal. Any remaining bits in the packet are considered ancillary data.

Referring again to Figure 1, the CODEC 1 provides for transmission of ancillary data via an asynchronous, bi-directional RS-232 input interface 39, and an output interface 62. These interfaces provide a transparent channel for the transmission of 8 data bits. The data format is 1 start bit, 8 data bits, 1 stop bit and no parity bits. A maximum data rate might be selected by the

control processor 5. This interface is capable of transmitting at the maximum data rate selected for the encoder 1 and the decoder 2 and thus no data pacing such as XON/XOFF or CTS/RTS are provided.

The RS-232 data rates might be set from 3 to 19,2 bps. The use of the ancillary data channel decreases the number of bits available to the audio channel. The reduction of the audio bits only occurs if ancillary data is actually present. The data rate might be thought of as a maximum data rate and if there is no ancillary data present, then no ancillary data bits are transmitted. A typical example of this situation occurs when the CODEC 1 is connected to a terminal; when the user types a character the character is sent to the decoder at the bit rate specified.

The setting of the decoder baud rate selection dip switches is done by considering the setting of the encoder. The decoder baud rate must be an equal or higher baud rate relative to the encoder. For example, it is possible to set the decoder ancillary baud rate to 9,6 baud. In this case, the encoder baud rate may be set to any value from 3 to 9,6 but not 19,2. If the decoder baud rate is set to a higher rate than the encoder, the data will burst out at the decoder's baud rate. The maximum sustained baud rate is therefore controlled by the encoder.

The compression technique for the transmission of ancillary data is as follows: the encoder looks, during each 24 millisecond frame interval, to see if any ancillary data is in its input

buffer. If there are characters in the encoder's input buffer, then the maximum number of characters consistent with the selected baud rate are sent. During a 24 millisecond period, the table below shows the maximum number of characters per frame (at 48 kHz sampling rate) sent for each baud rate.

BIT RATE	NUMBER OF CHARACTERS
3	1
12	3
24	6
36	9
48	12
72	18
96	24
192	47

The CODEC 1 provides no error detection or correction for the ancillary data. The user assumes the responsibility for the error control strategy of this data. For example, at an error rate of 1e-5 (which is relatively high) and an ancillary data rate of 12 baud, 1 out of every 83 characters will be received in error. Standard computer data communication protocol techniques might be used to maintain data integrity. When designing an error protection strategy, it must be remembered that the CODEC 1 may occasionally repeat the last 24 milliseconds of audio under certain error conditions. The effect on audio is nearly imperceptible. However, the ancillary data is not repeated.

The format of the ancillary data is user defined. The present invention utilizes two formats for the ancillary data. The first format treats the entire data stream as one logical (and physical) stream of data. The second format allows for multiplexing of

various logical and diverse data streams into one physical data stream. For example, switch closure, RS232, and time code data are all multiplexed into a single physical data stream and placed in the ancillary data stream of the ISO MPEG packet.

Figure 1 shows a high level diagram of the asynchronous multiplexer (MUX) 3 in relation to the other CODEC components. Figure 3 shows an isolated diagram of the multiplexer 3 in relation to encoder 1. The data rate for the multiplexer is set by software command (via remote control connections or keypad entry). A software command also controls a switch 34 (Figure 1) which routes the ancillary data through multiplexer 3. Multiplexer output line 36 routes the multiplexed data into the encoder input line 38. Alternatively, if the switch 34 is in the other position, ancillary data will be routed directly to the encoder input line 38 via the input line 32 without multiplexing. When the multiplexer 3 is used, Figure 1 shows signals from input sources such as RS485 (line 31), RS232 (line 33), contact closures (line 35), time codes (line 37), and ancillary data -- RS232 (line 39). Figure 3 shows similar inputs into multiplexer 3. These ancillary inputs are used as follows:

The RS232 I/O connector is used to provide an additional port into the data multiplexer. It might be thought of as a second RS232 ancillary port. The RS485 I/O connector is used to provide an additional type of port into the data multiplexer. It is a dedicated RS485 port and might be used to control RS485 equipment.

Contact closure inputs 3 allow simple ON/OFF switches to be interfaced into the CODEC 1. The contact closure inputs 3 are electrically isolated from the internal circuitry by optical isolators. A plurality of optical isolated I/O lines and/or contact closure lines might be used. Additionally, the time code inputs allow transmission of timecode at rates of 24, 25, 29, and 3 frames per second.

Referring again to Figure 3, the Ancillary Data Multiplexer 3 multiplexes the various inputs into a composite ancillary data stream for routing to encoder input line 38. The encoder 1 then processes the digital audio signals (e.g. converted left and right analog inputs, AES/EBU, SPDIF, or optical) and the ancillary data stream (e.g. multiplexed composite or direct) into a compressed audio bit stream. In Figure 3, an ISO/MPEG encoder 1 is shown, with the digital audio left and right signals, as well as a composite ancillary data stream, being processed by the ISO/MPEG encoder 1 into a resulting ISO/MPEG audio bit stream. Other compression techniques besides ISO/MPEG could similarly be illustrated.

Conversely, a block diagram is shown in Figure 4 wherein the ISO/MPEG Audio Bit Stream enters an ISO MPEG Decoder 2 on line 22. The bit stream is decoded (decompressed) and the ancillary data is separated from the audio data. The composite ancillary data stream enters the Ancillary Data De-Multiplexer 6 through line 23. The Ancillary data is de-multiplexed into its component parts of

Ancillary, RS232, RS485, Time Code, and Relay Contact data, as shown by lines 61, 63, 65, 67, and 69. The audio data (left and right) is output on lines 26 and 28. A software command also controls a switch 64 (Figure 1) that might route the ancillary data out of decoder 2, through the de-multiplexer 6, through line 66, and out to ancillary data line 69. Alternatively, the ancillary data might be routed directly from decoder output line 23, though line 62, and out line 69 -- without multiplexing.

Referring again to Figure 2, this more detailed block diagram of CODEC 1 shows external ancillary data entering the ancillary data switch 16 via line 39 and exiting switch 16 via line 69. (See lines 39, 69 and switches 34, 64 in Figure 1). Switch 16 interacts with Ancillary Data UART (Universal Asynchronous Receiver Transmitter) via connections 164 and 165. Switch 16 also interacts with DSP Ancillary Data UART 169 via connections 166 and 167. The resulting data is sent through Switch 16 to encoder 1 via connection 162. Decoded ancillary data is sent through Switch 16 from decoder 2 via connection 163. Switch 16, Ancillary Data UART 168, and DSP Ancillary Data UART are controlled by Control Micro 5 via control network line 155.

Figure 2 also details the following ancillary data connections: External RS232 data is shown entering RS232 UART 17 via line 33 and exiting UART 17 via line 69. External Time Code Data is shown entering SMPTE Time Code Interface 172 via line 37 and exiting via line 67. Time Code Data is subsequently shown

interacting with Time Code UART 174 via lines 173, 175. External RS485 data is shown entering RS485 UART 176 via line 31 and exiting via line 61. External Optical inputs are shown entering Control micro network 155 via line 35. Relay outputs are shown exiting Control micro network 155 via line 65. UARTS 17, 174, 176, and Time Code Interface 172 are controlled by Control Micro 5 via control network line 155.

Ancillary data can prove to be extremely valuable because it allows the CODEC user to transmit control and message information to and from RS232 and RS485 equipment, on either end of the transmission channel, via the same compressed digital bit stream as used by the audio signal component. The user might also send time code information and facilitate the control of relay contacts. More importantly, the use of ancillary data does not adversely affect the ability to transmit a sufficiently large amount of primary audio data.

Synchronous Ancillary Data

Referring again to Figure 1, the CODEC 1 also provides a synchronous ancillary input data line 18 and output data line 25. The synchronous connections might exist separately (as shown in Figures 1 and 2) or as part of a multi-functional input line (e.g. optical isolated I/O, relay I/O and synchronous ancillary data I/O share a common line -- not shown). This data port is an RS232 interface, and might also include RS422 and/or RS485 capabilities.

Digital Interface Modules and Loopback Control

Referring again to Figure 1, encoder 2 outputs a compressed audio bit stream through line 4 (and possibly more lines) into at least one DIM 7. These modules might include, for example, the types X.21/RS422, V.35, and/or TA. These modules output the digital signals for use and/or transmission by equipment external to the CODEC. Similarly, DIM 8 is connected to decoder 2 through line 81. DIM 8, using similar type modules as DIM 7, collects the external digital signals for transmission to decoder 2.

Referring again to Figure 2, this more detailed block diagram of CODEC 1 shows the compressed bit stream entering H.221 DSP 19 via line 191. DSP processes the bit stream and transfers the data, via line 192, to at least one DIM (Module types shown as 198). DIM 198 interacts with TA Control UART 193 via lines 194, 195, and with Decoder DSP 2 via line 197. DIM 192 then outputs external data via line 71 and inputs external data via line 81. As discussed above, this external data is then used by external equipment such as transmitters and receivers.

Before any connection is made to the outside world, the DIMs in CODEC 1 must be defined. If the DIMs are rearranged, then the CODEC must be notified via remote control software command (through the keypad or remote control interface). For DIMs that dial outside networks, two methods of dialing exist. They are single line dialing and multiple line dialing (speed dialing). For either mode of dialing it is possible to enable automatic reconnect. This

feature allows the automatic reconnection of a dropped line. If auto reconnect is enabled when a line is dialed, then it will be reconnected if either the far end disconnected the call, or the network drops the call. If the calling end drops the call, the line will not be automatically reconnected. This feature also allows the DIM to automatically dial an ISDN network if, for instance, a satellite connection is lost.

The CODEC 1 provides for two types of loopback through loopback control module 9. Loopback is an important feature for CODEC testing purposes. The first type is a system loopback and the second is a digital interface loopback. The system loopback is an internal loopback which loops back all the digital interfaces and is set by one software command. The second type of loopback allows the user to select individual digital interface modules for loopback. Loopback control might also be used to cause the internal CODEC clock to supply the digital data clocks.

Satellite Receiver Interfaced with CODEC

Referring to Figure 13, another embodiment of the disclosed invention allows for the transmission of other information besides audio, including, video, text, and graphics. In this embodiment, the digital line inputs 41 are preferably replaced with a satellite antenna 46. The digital interface module 42 (or satellite receiver module) receives digital signals that are transmitted to it by the satellite antenna 46. The digital signals, which are streams of

data bits, are then transferred to a decoder 42. The decoder decompresses the bits, whether they are audio, video, text, or graphic, and directs them to the appropriate output.

Preferably, the digital interface module 42 has the ability to store digital information. In this alternate embodiment, the digital interface module (satellite receiver module) is preferably a receiver called a "daX". Such a receiver is available commercially under the name "daX" from Virtual Express Communications in Reno, Nevada. In this embodiment, the decoder preferably would have the capability to decompress or decode other types of compressed information such as video, text, and graphics. This could be facilitated by downloading the required compression techniques into the CODEC 1 as described above.

In its operation, the satellite antenna 46 might receive digital information from various sources including a remote CODEC or a remote daX (not shown), and transfer the information to the daX receiver 42. The daX DIM 44 might also act as a switching mechanism to route the digital bit streams to different places. It might direct information received from the satellite directly to the decoder, via line 4, for decompression and immediate output. The received data from the satellite receiver 42 might alternatively be directed through the daX DIM 44 to the daX 45 via line 43 for storage and later retrieval. The digital interface module 44 might then direct these stored data bits from the daX 45 to the decoder 42 via path 4 for decoding and subsequent output.

This embodiment also preferably allows for simultaneous storage of digital information in the DAX via path 43 and for immediate decoding of digital information via line 4 through the decoder 42.

While few preferred embodiments of the invention have been described hereinabove, those of ordinary skill in the art will recognize that these embodiments may be modified and altered without departing from the central spirit and scope of the invention. Thus, the embodiments described hereinabove are to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims, rather than by the foregoing descriptions, and all changes which come within the meaning and range of equivalency of the claims are intended to be embraced herein.

CODEC SOFTWARE COMMAND DESCRIPTIONS AND CODEC OPERATIONS MANUAL

Attached hereto are relevant portions of the operation manual for the CODEC 1, which includes detailed descriptions of the remote control software commands and their usage with the CODEC 1 described above.

```
        opt     fc
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights
reserved.
;
; \UXCODE\bitalloc.asm:

; This routine is used to allocate the bits.
; It allocates at least some bits to all sub-bands with a positive
SMR.
; It allocates in three phases:
;     A. allocate all sub-bands until they are all below
;          the Global Masking Threshold (regardless as to how many
;          bits it takes)
;        note 1. a limit (sub-band boundary) is set which requires
;                all sub-bands up to the boundary require at least
;                index 1 be allocated even if the signal is already
;                below the Global Masking Threshold. (This provides
;                a noticeable improvement in continuity of sound)
;        note 2. For JOINT stereo framing,
;                a. a 1st pass is made thru Phase A with the frame type
;
;                   set to FULL stereo to see if the framing bit pool
;                   can handle FULL stereo.
;                   The 1st sub-band (channel) that exceeds the bit pool
as
;                   it is allocated for below the Global Maksing
Threshold
;                   causes the 1st pass thru Phase A to be aborted
;                   indicating the a JOINT frame is necessary.
;                b. JOINT framing uses the aborted sub-band to set the
;                   intensity sub-band boundary to 4, 8, 12 or 16.
;                   A new bit pool is determined based on this boundary.
;                   A call to the routines to calculate the JOINT Stereo
;                   arrays is made.
;                   A 2nd pass thru Phase A is made for a JOINT stereo
;                   frame and a new bit pool size.
;     After Phase A is completed, a test is made to see if the bit
pool
;          was overflowed by the allocation.
;     a. if the frame fits, Phase B is skipped and Phase C is done
;     b. otherwise, Phase B is required to selectively de-allocate
the
;          best sub-band candidates.
;
;
; on entry
;     y:<stereo = flags:
;     (set on entry) bit 0 means stereo vs mono framing
;                                0 = stereo framing
;                       1 = mono framing
;                bit 1 is used to indicate left vs right channel
;                                0 = looping through left channel
arrays
;                       1 = looping through right channel arrays
;
```


BAD ORIGINAL

```
;                 bit 2 is to simply indicate that joint stereo applies
;                                  0 = NOT joint stereo framing type
;                                  1 = IS joint stereo framing type
;                 bit 3 is to indicate the full stereo initial
allocation
;                  if joint stereo applies
;                                  0 = normal joint stereo allocation
;                                  1 = FULL STEREO initial joint stereo
allocation
;                 bit 4 is to simply indicate the stereo intensity
sub-band
;                   boundary has been reached if joint stereo applies
;                                  0 = NO sub-bands still below
intensity boundary
;                                  1 = sub-bands above intensity
boundary
;                 bit 5 is used as the FirstTime switch in an
allocation
;                                  0 = cleared if any allocations were
made
;                                  1 = no allocations made to any
sub-bands
;                 bit 6 is used for critical de-allocate and allocate
passes:
;                       with below masking threshold being a criteria
;                  de-allocate:
;                       0 = select from any sub-band channel
;                       1 = select from only those below mask
;                  allocate:
;                                  0 = there are sub-band channels not
below mask
;                                  1 = all sub-bands are below mask
;                 bit 7 is used for critical de-allocate and allocate
passes:
;                  de-allocate:
;                       0 = select from any sub-band channel
;                       1 = select from those with 2 or more allocation
;                  allocate:
;                                  0 = are sub-bands not below hearing
thresh
;                                  1 = all sub-bands are below hearing
thresh
;                 bit 8 is used for critical de-allocate and allocate
passes:
;                  de-allocate:
;                       0 = select from any sub-band channel
;                       1 = select from any sub-band channel
;                  allocate: for final pass after bit allocation timer
;                                  0 = timer interrupt not yet sensed
;                                  1 = timer interrupt was sensed
;                 bit 9 is to simply indicate that the sub-band limit
```

```
for
;                allocating at least ONE position has been reached
;                within a current loop:
;                                  0 = NOT at sub-band limit
;                                  1 = reached the sub-band limit
;               bit 10 is to simply indicate that the maximum
sub-band for
;                consideration for allocation has been reached
;                within a current loop:
;                                  0 = NOT at maximum sub-band limit
;                                  1 = reached the maximum sub-band
limit
;
;     y:audbits = number of bits available for sbits, scale factors
and data
;     y:<usedsb  = number of sub-bands actually used
;     y:<maxsubs = MAXSUBBANDS at sampling rate and bit rate
;     y:sibound = stereo intensity sub-band boundary
;     y:stintns = stereo intensity sub-band boundary code for frame
header
;     y:limitsb  = number of sub-bands requiring at least one
allocation
;     y:bandcnt  = decremented sub-band counter intensity boundary
check
;     y:frmtype = framing type specified by external dip switches
;     y:opfrtyp = current frame output type (Joint may upgrade frame
to full)
;     y:<qtalloc = timer interrupt set to signal quit allocation
loops
;     r0 = addr of the SBits array left and right channel (x memory)
;     r1 = addr of MinMasking Db array left and right channel (x
memory)
;     r2 = addr of SubBandMax array left and right channel (x
memory)
;     r4 = addr of the SubBandPosition array left and right channel
(x memory)
;     r5 = addr of the SubBandIndex array left and right channel (x
memory)
;
; on exit
;     a = destroyed
;     b = destroyed
;     x0 = destroyed
;     x1 = destroyed
;     y0 = destroyed
;     y1 = destroyed
;     r3 = destroyed
;     r6 = destroyed
;     n0 = destroyed
;     n1 = destroyed
;     n2 = destroyed
;     n3 = destroyed
;     n4 = destroyed
;     n5 = destroyed
```

```
;    n6 = destroyed
;
;   AtLimit array by sub-bands (left for 32, then right for 32):
;         bit 0 set when allocation is below the masking threshold
;         bit 1 set when allocation is below the threshold of
hearing
;         bit 2 set when allocation is at the limit of maximum
position
;                or there are not enough bits to allocate
;                the sub-band further

     include   'def.asm'
     include   'box_ctl.asm'

     section   lowmisc

     xdef BitsAdd
     xdef BPosAdd
     xdef BInxAdd
     xdef AllwAdd
     xdef MNRsub
     xdef MNRsb
     xdef MNRchan
     xdef MNRmin
     xdef MNRinx
     xdef MNRpos,
     xdef AvlBits
     xdef TotBits
     xdef HldBits
     xdef count
     xdef svereg
     xdef jntflag

     org  yli:
stbitalloc_yli

BitsAdd   ds   1          ;save address of SBits array
BPosAdd   ds   1          ;save address of SBPosition array
BInxAdd   ds   1          ;save address of SBIndex array
AllwAdd   ds   1          ;save addr of applicable Allowed
table
MNRsub    ds   1          ;count of entries in de-allocate
tables
MNRsb     ds   1          ;curr sub-band for allocation
MNRchan   ds   1          ;channel of curr sub-band for
allocation
MNRmin    ds   1          ;value of curr sub-band for
allocation
MNRinx    ds   1          ;new index for selected sub-band
MNRpos    ds   1          ;new allowed position for selected sb
AvlBits   ds   1          ;available bits to allocate
TotBits   ds   1          ;current bit count allocated
HldBits   ds   1          ;sub-band critical allocation
count     ds   1          ;sub-band counter
```

```
svereg     ds    1                   ;save register for restoring
jntflag    ds    1                   ;bits to control joint functions:
                                  ;  0 - for demand bits pass
                                  ;       0 - not at available bits yet
                                  ;       1 - reached end of avail bits

endbitalloc_yli
        endsec

        section highmisc

        xdef    UsedSBs
     xdef bitallocR7Save
     xdef bitallocN7Save
     xdef bitallocM7Save

        org     xhe:
stbitalloc_xhe

;This array is the counters for sub-bands with assigned indices
;If a sub-band starts out below the Global Masking Threshold it
takes
;a certain number of consecutive frames before it is skipped. Until
that
;count down (SUBBANDSCTDOWN) reaches zero, the sub-band will
receive at
;least one allocation.

UsedSBs   ds   NUMSUBBANDS*2

;these save variables for exclusive use by bitalloc only

bitallocR7Save ds   1
bitallocN7Save ds   1
bitallocM7Save ds   1

endbitalloc_xhe
     endsec

     section   highmisc
     xdef strtsin
     xdef endsin
     xdef uselmsb
     xdef demand
     xdef jntadj
     xdef jntsub
     xdef boundlst
     xdef isocdelst
     xdef jntfrms
     xdef jfrmcnt
     xdef UsedSBReg
     xdef MaxPos
     xdef ndatabit
        xdef    NDataBit
```


BAD ORIGINAL

```
      xdef NSKFBits
      xdef SNR

      org  yhe:
stbitalloc_yhe

;sub-band range for a possible sine wave in the current channel
;if not a sine wave, these values are -1

strtsin    ds    1          ;start sub-band span for sine wave
endsin     ds    1          ;end sub-band span for sine wave
uselmsb    ds    1          ;use  LIMITSUBBANDS  not  greater  thane
y:<usedsb
demand     ds    1          ;demand bits
jntadj     ds    1
jntsub     ds    1
boundlst   ds    1     ;boundary to use for jntfrms
isocdelst ds     1     ;intensity boundary ISO code for boundlst
jntfrms         ds    1     ;count of frames to maintain
jfrmcnt         ds    1     ;frame counter
UsedSBReg ds     1     ;current addr into UsedSbs counters array
MaxPos          ds    1     ;Max Position per selected Allowed table

;This is the addr of the selected table, ISO or COMPRESS,
;    for the number of bits for data allocation by position

ndatabit  ds    1          ;addr of ISO or COMPRESS NDataBit tbl

;This is the ISO table for the number of bits for data allocation
by position

NDataBit
        dc      0*NUMPERSUBBAND        ;index = 0, no transmit =
0   bits
        dc      5*NUMPERSUBBAND        ;index = 1, packed      =
60  bits
        dc      7*NUMPERSUBBAND        ;index = 2, packed      =
84  bits
        dc      9*NUMPERSUBBAND        ;index = 3              =
108 bits
        dc      10*NUMPERSUBBAND       ;index = 4, packed      =
120 bits
        dc      12*NUMPERSUBBAND       ;index = 5              =
144 bits
        dc      15*NUMPERSUBBAND       ;index = 6              =
180 bits
        dc      18*NUMPERSUBBAND       ;index = 7              =
216 bits
        dc      21*NUMPERSUBBAND       ;index = 8              =
252 bits
        dc      24*NUMPERSUBBAND       ;index = 9              =
288 bits
        dc      27*NUMPERSUBBAND       ;index = 10             =
324 bits
```

```
        dc      30*NUMPERSUBBAND        ;index = 11             =
360 bits
        dc      33*NUMPERSUBBAND        ;index = 12             =
396 bits
        dc      36*NUMPERSUBBAND        ;index = 13             =
432 bits
        dc      39*NUMPERSUBBAND        ;index = 14             =
468 bits
        dc      42*NUMPERSUBBAND        ;index = 15             =
504 bits
        dc      45*NUMPERSUBBAND        ;index = 16             =
540 bits
        dc      48*NUMPERSUBBAND        ;index = 17             =
576 bits

;This is the COMPRESS table for number of bits for data allocation
by position

        dc      0*NUMPERSUBBAND         ;index = 0, no transmit =
0   bits
        dc      4*NUMPERSUBBAND         ;index = 1, packed      =
48  bits
        dc      6*NUMPERSUBBAND         ;index = 2, packed      =
72  bits
        dc      8*NUMPERSUBBAND         ;index = 3, packed      =
96 bits
        dc      10*NUMPERSUBBAND        ;index = 4, packed      =
120 bits
        dc      12*NUMPERSUBBAND        ;index = 5              =
144 bits
        dc      15*NUMPERSUBBAND        ;index = 6              =
180 bits
        dc      18*NUMPERSUBBAND        ;index = 7              =
216 bits
        dc      21*NUMPERSUBBAND        ;index = 8              =
252 bits
        dc      24*NUMPERSUBBAND        ;index = 9              =
288 bits
        dc      27*NUMPERSUBBAND        ;index = 10             =
324 bits
        dc      30*NUMPERSUBBAND        ;index = 11             =
360 bits
        dc      33*NUMPERSUBBAND        ;index = 12             =
396 bits
        dc      36*NUMPERSUBBAND        ;index = 13             =
432 bits
        dc      39*NUMPERSUBBAND        ;index = 14             =
468 bits
        dc      42*NUMPERSUBBAND        ;index = 15             =
504 bits
        dc      45*NUMPERSUBBAND        ;index = 16             =
540 bits
        dc      48*NUMPERSUBBAND        ;index = 17             =
576 bits
```

```
;Each sub-band, if it is transmitted, must send scale factors.  The
;Sbit patterns determine how many different scale factors are
transmitted.
;The number of scale factors transmitted may be 0, 1, 2 or 3.  Each
scale
;factor requires 6 bits.
;
;Sbit patterns
;       00      Transmit all three scale factors          18 (3 * 6
bits)
;       01      Transmit the second two scale factors     12 (2 * 6
bits)
;       10      Transmit only one scale factor             6 (1 * 6
bits)
;       11      Transmit the first two scale factors      12 (2 * 6
bits)

;The NBits array is used to determine the number of bits to
allocate for the
;scale factors. NSBITS (the 2 bits for SBits code) are added to
account for
;all required scale factor bits (18+2,12+2,6+2,12+2).

NSKFBits
        dc      20,14,8,14

;This is the table for Signal to Noise ratio by position

        include '..\x1code\snr.asm'

endbitalloc_yhe
     endsec

        org     phe:

bitalloc

;    bset WATCH_DOG        ;tickle the dog
;    OFF_BITALLOC_LED_CD        ;tickle the led

;save register 7 and its attendants

     move r7,x:bitallocR7Save
     move n7,x:bitallocN7Save
     move m7,x:bitallocM7Save

     move #-1,m7                ;set to a linear buffer control

;Save the left and right channel array starting addresses

        move    r0,y:BitsAdd                ;save register of SBits
array
        move     r4,y:BPosAdd                    ;save register of
```


BAD ORIGINAL

```
SubBandPosition array
        move        r5,y:SInxAdd                         ;save register of
SubBandIndex array

;select the ISO or COMPRESS table for NDataBit:

     move #NDataBit,r5          ;standard ISO table
     move #18,n5                ;offset to COMPRESS table
     jclr #USE_COMPRESS,y:<cmprsctl,_bita_05_A
     move (r5)+n5               ;select the COMPRESS table

_bita_05_A
     move r5,y:ndatabit         ;set addr of NDataBit table for alloc

;set up the MNR arrays for the left and right channels and the
joint channel
; if applicable

     move #SBMsr,r5        ;addr of Mask-to-Signal by sub-band
     move #NUMSUBBANDS,n5          ;offset to right channel values
     move r5,r3                 ;addr of left chan Mask-to-Sig array
     move (r5)+n5               ;add offset to right channel
     move r5,r4                 ;addr of right chan Mask-to-Sig array
     move (r5)+n5               ;add 2nd offset to joint channel
     move n5,n1                 ;access right channel MinMsk values
     move n5,n2                 ;access right channel SBMax values

;apply the safety factor

     move y:o_psych,y0          ;get the safety factor

;loop through the required sub-bands

     do   y:<usedsb,_bita_30_A
     move x:(r2+n2),x0          ;get right channel SBMax
     move x:(r1+n1),b           ;get right channel MinMsk
     sub  x0,b x:(r2)+,x0       ;MinMask  -  SBMax  =  Mask-to-Signal
ratio
                           ; & get left channel SBMax, incr nxt sb
     sub  y0,b x:(r1)+,a   ;apply safety factor to right channel
                           ; & get left channel MinMsk, incr nxt sb
     move b,x:(r4)+        ;store for test if below mask already
     sub  x0,a             ;MinMask - SBMax = Mask-to-Signal ratio
     sub  y0,a             ;apply safety factor to left channel
     move a,x:(r3)+        ;store for test if below mask already

;if doing joint stereo, develop the Joint Mask-to-Signal from the
lesser
;    of the left and right channels

     jclr #JOINT_FRAMING,y:<stereo,_bita_20_A
     cmp  a,b              ;compare left and right MNR values
     jlt  <_bita_10_A            ;b (right chan) is less, store that
one
```

```
     move a,x:(r5)+          ;otherwise store a (left chan) as less
     jmp  <_bita_20_A

_bita_10_A
     move b,x:(r5)+          ;b (right chan) is less, store that one

_bita_20_A
     nop

_bita_30_A                          ;END of y:<usedsb do loop

;set the working value for bits available for allocation
; NOTE: this value may be changed for JOINT stereo if the FULL
stereo
;        bit allocation for the frame CANNOT be handled
;..      (for JOINT stereo,
;          y:audbits is the available bit count for FULL stereo)

     move y:audbits,x0               ;get standard available bit cnt
     move x0,y:AvlBits               ;store as working bit cnt

;save original array of used sub-band count down counters

     move #UsedSBs,r0
     move #SvUsedSBs,r1
     do   #NUMSUBBANDS*2,_bita_31_A
     move x:(r0)+,x0
     move x0,x:(r1)+

_bita_31_A

;initialize the bit allocation control flags in y:<stereo

     bclr #JOINT_at_FULL,y:<stereo ;init flag NOT at FULL

;if doing joint stereo,
; set flag for initial allocation to drive subbands to masking
;    threshold to see if frame can handle full stereo

     jclr #JOINT_FRAMING,y:<stereo,_bita_40_A  ;not joint frames,
continue
     bset #JOINT_at_FULL,y:<stereo ;set for initial Joint pass
     move #0,x1                     ;clear joint flag
     move x1,y:<jntflag             ;for joint demand bit rate ctl

_bita_40_A

;set usable LIMITSUBBANDS: if greater than y:<usedsb, use y:<usedsb

     move y:limitsb,x1              ;get static LIMITSUBBANDS
     move y:<usedsb,a               ;get the used sub-band cnt
     cmp  x1,a x1,y:uselmsb         ;test limitsb vs usedsb
                               ; & in case, set usable limistb
     jge  <_bita_41_A               ;if used > limit, continue
```

```
;used sub-band count is less the LIMITSUBBANDS, set to used
sub-bands

     move a,y:uselmsb

_bita_41_A

;
;(c)  TotBits = 0;              /* start the bit allocation counter
*/
;
        clr    a    #>1,x1         ;total bit used, x1 = 1 for
start index
     move a,y1            ;y1 = 0 to initialize
     move a,y:TotBits
     move a,y:count       ;start the sub-band counter
     bclr #AT_LIMIT_SUBBAND,y:<stereo   ;NOT yet at sub-band limit
                          ; which require at least 1 allocation
     bclr #AT_USED_SUBBAND,y:<stereo    ;NOT  yet  at  sub-band
maximum
                          ; limit for coding used sub-bands

;
;initial allocation for all sub-bands;
;    1. that are within the use (less than UsedSubBands
;    2. with a MinimumMasking to MaximumSignal above the masking
threshold
;
     move #SBMNRmax,r0          ;addr of de-alloc Max signal-noise
     move #SBMsr,r1        ;addr of Mask-to-Signal by sub-band
        move    y:BitsAdd,r2        ;set register of SBits array
        move    y:AllwAdd,n3        ;init the current Allowed table
        move    y:BPosAdd,r4        ;set register of SubBandPosition
array
        move    y:BInxAdd,r5        ;set  register  of  SubBandIndex
array
     move #UsedSBs,r6           ;set start addr of used sub-band cnts
     move r6,y:UsedSBReg        ;set current (0) used sub-band cnt
addr
        move    #AtLimit,r6         ;point to SubBandAtLimit array

;in case of joint stereo, clear the reached intensity sub-band
boundary flag

     move y:sibound,x0          ;joint stereo intensity sub-band
     move x0,y:bandcnt          ;bound subband decremented cntr
     bclr #JOINT_at_SB_BOUND,y:<stereo  ;clear  reached  boundary
sub-band

;initial allocation pass
;do all required sub-bands alternating between the left and right
channels
; for the joint stereo 2nd pass make address alterations for joint
arrays
```

```
     do    #NUMSUBBANDS,_bita_230_A

;clear the n registers for the left channel reference

     clr  a    #0,n0          ;clear reg a to zero
                         ; & set n0 for left channel SBMNRmax
     move a,n1           ;SBMsr array
     move a,n2           ;SBits or Joint SBits array
     move a,n4           ;SBPos array
     move a,n5           ;SBIndx array
     move a,n6           ;AtLimit array
     bclr #LEFT_vs_RIGHT,y:<stereo   ;flag for left channel in
progress

;initialize for the possible presence of a sine wave in left
channel
;get the left xpsycho sine wave sub-bands to handle possible sine
wave

     move y:strtsinlft,a      ;start entry equals 1st sub-band
     move a,y:strtsin         ;isolate the starting sub-band
     move y:endsinlft,a       ;end entry equals last sub-band
     move a,y:endsin          ;isolate the ending sub-band

; if joint stereo does NOT apply, continue

     jclr #JOINT_FRAMING,y:<stereo,_bita_60_A

; if joint stereo upgraded to full, continue

     jset #JOINT_at_FULL,y:<stereo,_bita_60_A

; if doing joint stereo and have already switched over to joint
SBits array,
;    but now have to adjust to 3rd set of SBMsr values

     jset #JOINT_at_SB_BOUND,y:<stereo,_bita_50_A

; see if the joint stereo intensity sub-boundary has been reached
; if not, continue at full stereo for these early sub-bands
; otherwise, switch over to the JointSBits

     move y:bandcnt,r3        ;get decrement sub-band ctr
     jsr  chkjoint       ;see if reached boundary
     move r3,y:bandcnt        ;save new decremented ctr
     jclr #JOINT_at_SB_BOUND,y:<stereo,_bita_60_A

     move #JntSBits,r2        ;shift over to Joint SBits array
     move y:count,n2          ;to offset to current sub-band
     nop
     move (r2)+n2             ;adj addr to current sub-band
     move #0,n2               ;reset to left channel

_bita_50_A
```

```
;we're at intensity sub-band limit
; shift over to Joint channel in SBMsr array
;    (3rd set of sub-band values in n1)

     move #NUMSUBBANDS*2,n1          ;Joint SBMsr values by sub-band

_bita_60_A

;process the current channel

     do   #NUMCHANNELS,_bita_220_A

;initialize the pertinent sub-band values to 0

     move y1,x:(r6+n6)          ;clear allocated limit flag (AtLimit)
     move y1,x:(r5+n5)          ;clear allocated index (SBIndx)
     move y1,x:(r4+n4)          ;clear allocated position (SBPos)

;if we reached the used sub-band limit,
;  take this one out of the picture completely

     jset #AT_USED_SUBBAND,y:<stereo,_bita_185_A

;if doing mono and we are processing the right channel,
;  take this one out of the picture completely

     jclr #STEREO_vs_MONO,y:<stereo,_bita_70_A ;if doing stereo,
continue
     jset #LEFT_vs_RIGHT,y:<stereo,_bita_185_A ;if right, bag this
one

_bita_70_A
     jclr #JOINT_at_SB_BOUND,y:<stereo,_bita_80_A
     jset #LEFT_vs_RIGHT,y:<stereo,_bita_185_A   ;right chan at
intensity                            ; take sub-band out of picture totally

_bita_80_A
     move y:count,y0            ;get current sub-band (00-31)

;see if we reached the used sub-band limit

     jset #LEFT_vs_RIGHT,y:<stereo,_bita_85_A   ;left channel did
this
     move y:<usedsb,b           ;get count of used subbands for
testing
     cmp  y0,b             ;see if sub-band not to be coded
     jgt  <_bita_85_A           ;if not, continue
     bset #AT_USED_SUBBAND,y:<stereo     ;just reached sub-band
maximum
     jmp  <_bita_185_A          ;take completely out of use

_bita_85_A
```

```
;save current sub-band AtLimit addr for re-use after UsedSBs
counter processed
; and set address of current sub-band in use count down counter

     move r6,y:svereg
     move y:UsedSBReg,r6

;if we reached the sub-band limit for those requiring at least one
sub-band,
;  see if we have anything to allocate to get below the Global
Masking Threshold

     jset #AT_LIMIT_SUBBAND,y:<stereo,_bita_90_A

;see if at least one allocation is required regardless of signal to
noise ratio

     jset #LEFT_vs_RIGHT,y:<stereo,_bita_95_A   ;left channel did
this
     move y:uselmsb,a          ;get sub-band limit for at least 1
alloc
     cmp  y0,a            ;if there is initial allocation
     jgt  <_bita_95_A          ;continue
     bset #AT_LIMIT_SUBBAND,y:<stereo   ;just reached that limit

_bita_90_A

;if this channel has a sine wave, continue the allocation algorithm

     move y:strtsin,a          ;get start sub-band if sine wave
     tst  a               ;if -1, no sine wave
     jge  <_bita_95_A          ;if NOT -1 it's sine wave, continue

;otherwise, see if below Mask-to-Signal

     move x:(r1+n1),a          ;get sub-band's Mask-to-Signal ratio
     tst  a     x:(r6+n6),a    ;test Mast-to-Sig for positive value
                          ; & get current count down value
     jle  <_bita_95_A          ;if  above  masking  thresh,  init
counter

;test the used sub-band count down counter to see if this sub-band
;  can be skipped from at least 1 allocation

     tst  a     y:svereg,r6    ;see if zero and can be skipped
                          ; & in case it can, reset AtLimit addr
     jle  <_bita_190_A         ;counter = zero, set Below Mask flag

;decrement the count down counter and make 1 allocation

     sub  x1,a y:UsedSBReg,r6 ;decrement
                          ; & set addr used sub-band counter
     jmp  <_bita_96_A          ;update count down counter
```

```
_bita_95_A

;initialize the used sub-band count down counter

;     move #>SUBBANDCNTDOWN,a
      move y:zl_psych,a          ;get the count down value

_bita_96_A

;update the used sub-band count down counter and reset AtLimit
address reg

      move a,x:(r6+n6)
      move y:svereg,r6

;look for a sine wave in this channel
; and if so,
;   see if the current sub-band is within the sine wave sub-band
range
;      and if so,
;           force the allocation to the maximum

      move y:strtsin,a           ;get start sub-band if sine wave
      tst  a    y:count,y0       ;if -1, no sine wave
                              ; & get current sub-band to test
      jlt  <_bita_97_A           ;if -1 not sine wave, continue
      cmp  y0,a y:endsin,a       ;current sub-band vs start sub-band
                              ; & get end sub-band
      jgt  <_bita_180_A          ;if not yet reached, do not allocate
      cmp  y0,a y:MaxPos,r7      ;current sub-band vs end sub-band
                              ; & set addr adj to max allocation
      jlt  <_bita_180_A          ;if passed, do not allocate
      bset #ALLOCATE_SINE,x:(r6+n6) ;flag sub-band as a sine wave
      jmp  <_bita_110_A          ;if in range, allocate the maximum

_bita_97_A

;otherwise,
;   find Signal-to-Noise position that puts Signal below Masking
Threshold

      move x1,r7                 ;start  at  1st  Signal-to-Noise
position
      move #SNR,n7               ;addr of Signal-to-Noise table
      move x:(r1+n1),y0          ;get signal to mask ratio

      dc   #NUMSNRPOSITIONS-1,_bita_110_A

      move y:(r7+n7),a           ;get the Signal-Noise at position
      add  y0,a             ;add MNR to SNR for test
      jle  <_bita_100_A          ;still above mask, try next position

;now below the Global Mask, quit the loop
```

```
     enddo                          ;found position, stop #NUMSNRPOS-1
loop
     jmp  <_bita_110_A              ;go to end of loop

_bita_100_A

; try the next position and continue the loop

     move (r7)-                     ;try next Sig-Noise position

_bita_110_A                         ;END of #NUMSNRPOSITIONS-1 do loop

     move r7,y0                     ;save the matched SNR position
     move y:MaxPos,a                ;to test if exceeded max position
     cmp  y0,a y1,r3                ;is counted pos greater than max
                               ; & start at index 0 with allocation
     jge  <_bita_115_A              ;if not, go on to match the index
     move a1,y0                     ;set position at the maximum

_bita_115_A

;find index of the position that best matches the selected SNR
position

     do    #NUMINDEXES,_bita_130_A

     move x:(r3+n3),a               ;get the sub-band indexed position
     cmp  y0,a                 ;compare to selected position
     jlt  <_bita_120_A              ;match not found yet, try next index

;found the matching index, quit the loop

     enddo                          ;found index, stop #NUMINDEXES loop

;   a. if doing a sine wave in this sub-band, accept maximum
position index
;  b. otherwise, see if maximum position assigned and if so,
;     back up one index to the next to last index for this sub-band

     jset #ALLOCATE_SINE,x:(r6+n6),_bita_130_A    ;if sine, accept
index
     move y:MaxPos,y0               ;max position for Allowed table
selected
     cmp  y0,a                 ;see if max position assigned
     jlt  <_bita_130_A              ;if not, accept the assigned index
     move (r3)-                     ;back up to the next-to-last index
     move x:(r3+n3),a               ;assign the next-to-last index
position
     jmp  <_bita_130_A              ;go to end of loop

_bita_120_A

;try the next index and continue the loop
```

```
     move  r3 -                  ;try position at next index

;see if end of the table line reached

     move x:(r3+n3),a            ;get this next index to test
     tst  a                 ;test for an index of zero
     jne  <_bita_125_A           ;if not 0, keep looking

;index of zero indicates no higher indices apply, back up 1 and use
that

     move (r3)-                  ;use previous index
        bset      #ALLOCATE_LIMIT,x:(r6+n6)  ;set the completely
allocated bit
        bset      #HEARING_LIMIT,x:(r6+n6)   ;set the completely
allocated bit
     move x:(r3+n3),a            ;assign the last index position
     enddo                       ;found index, stop #NUMINDEXES loop
     jmp  <_bita_130_A           ;go to end of loop

_bita_125_A
     nop                    ;keep looping

_bita_130_A                      ;END of #NUMINDEXES do loop

;set the initial allocation SubBandIndex and SubBandPosition

     move r3,x:(r5+n5)           ;set initial allocation SBIndx
     move a1,x:(r4+n4)           ;set initial allocation SBPos

;determine the number of scale factor bits allocated at this
position

     move x:(r2+n2),n7           ;get the SBits scale factor code
(0-3)
     move #NSKFBits,r7           ;addr SBits scale factor bit count
tbl
     nop
     move y:(r7+n7),y0           ;save the scale factor bit count

;if joint stereo and we have reached the intensity sub-band
boundary
;    add the right channel joint SBits bit count also

     jclr #JOINT_at_SB_BOUND,y:<stereo,_bita_140_A
     move #NUMSUBBANDS,n2              ;offset to right channel Joint
SBits
     nop
     move x:(r2+n2),n7           ;get the SBits scale factor code
(0-3)
     move #0,n2                  ;restore to left channel Joint SBits
     move y:(r7+n7),a            ;save the scale factor bit count
     add  y0,a              ;add left to right Joint SBits cnt
     move a,y0              ;restore to proper register
```

```
_bita_140_A

;add the bits required for the signal data

     move x:(r4+n4),n7          ;get the position
     move y:ndatabit,r7         ;addr of NDataBit count by position
     nop
     move y:(r7+n7),a           ;get the bit count at this position
     add  y0,a y:TotBits,x0     ;add scale factor bits
                           ; & get curr TotBits
     add  x0,a y:AvlBits,x0     ;update  TotBits  with  bits  just
allocated
                           ; & get available bits
     move a,y:TotBits           ;save new allocated total bits

; if joint stereo run at full, see if total available bits exceeded

     jclr #JOINT_at_FULL,y:<stereo,_bita_150_A
     cmp  x0,a             ;check if room for allocation
     jle  <_bita_150_A          ;if room, continue

; not enough room for FULL stereo, we have to do Joint Stereo
; if already joint was sensed, continue developing demand bit rate

     jset #0,y:<jntflag,_bita_150_A     ;joint   sensed   before,
continue

;1st indication of joint:
;    indicate we found joint is needed
;    save the sub-band number at this point

     bset #0,y:<jntflag         ;indicate joint sensed
     move y:count,a        ;get the sub-band number
     move a,y:jntsub            ;save the sub-band number for later

_bita_150_A

;check  that  Signal-to-Noise  position  that  Signal  below  Masking
Threshold

     move x:(r4+n4),n7          ;get the position
     move #SNR,r7               ;addr of Signal-to-Noise table
     move x:(r1+n1),y0          ;get signal to mask ratio
     move y:(r7+n7),a           ;get the Signal-Noise at position
     add  y0,a x:(r5+n5),r3     ;add MNR to SNR for test
                           ; & set up to set prev index for its pos
     jle  <_bita_160_A          ;above mask, skip next statement
     bset #MASKING_LIMIT,x:(r6+n6)   ;set  AtLimit  partially  done
allocate

_bita_160_A

;if joint stereo run at full, continue with the next channel
```

```
     ;set #JOINT_at_FULL,y:<stereo,_bita_200_A

;if a sine wave sub-band, fill out total allocation

     ;set #ALLOCATE_SINE,x:(r6+n6),_bita_185_A

;for others,
;set the value for testing the best sub-band to deallocate bits
from
;if the frame cannot handle the full required allocation

     move (r3)-                 ;back up one index to get that
position
     move x:(r3+n3),n7          ;get the position at the previous
index
     nop
     move y:(r7+n7),a           ;get the Signal-Noise at position
     add  y0,a             ;calc Sig-to-Noise at prev position
     move a,x:(r0+n0)           ;save in SBMNRmax array for later
     jmp  <_bita_200_A          ;continue with the next channel

_bita_180_A

;if channel has a sine wave, suppress any allocation during final
passes

     move y:strtsin,a           ;get start sub-band if sine wave
     tst  a     #>1,y0          ;if -1, no sine wave
                           ; & set up to test sub-band 1, if sine
     jlt  <_bita_185_A          ;if -1 not sine wave, continue

;for current sub-bands 0 or 1 to suppress any allocation

     move y:count,b        ;get current sub-band (00-31)
     tst  b                ;check if sub-band 0, to suppress alloc
     jeq  <_bita_185_A          ;if 0, do not allocate
     cmp  y0,b             ;if sub-band 1, no allocate
     jeq  <_bita_185_A          ;if 0, do not allocate

;for 1st harmonic sub-bands (start and end times 2) to suppress any
allocate
;all other sub-bands are set as at masking limit to allow some
allocation
;of leftover bits

     asl  a                ;double start subband suppress harmonic
     cmp  a,b  y:endsin,a       ;see if current sub-band harmonic
                           ; & get set to test end subband harmonic
     jeq  <_bita_185_A          ;if harmonic, NO allocate
     asl  a                ;double end subband suppress harmonic
     cmp  a,b              ;see if current sub-band harmonic
     jne  <_bita_190_A          ;set as at masking limit
;    jeq  <_bita_185_A          ;if harmonic, NO allocate
```

```
_bita_185_A

;sub-band (channel) is not to be coded at all

     bset #ALLOCATE_LIMIT,x:(r6+n6)  ;set AtLimit totally out of
allocation
     bset #HEARING_LIMIT,x:(r6+n6)   ;set AtLimit at threshold of
hearing

_bita_190_A

;sub-band (channel) is set to indicate it is at its masking
threshold

     bset #MASKING_LIMIT,x:(r6+n6)   ;set AtLimit partially done
allocate

_bita_200_A

;finished the sub-band at the current channel
; a. if just finished the right, skip next instructions

     jclr #LEFT_vs_RIGHT,y:<stereo,_bita_210_A
     enddo                     ;to save cycles, stop #NUMCHANNELS
loop
     jmp  <_bita_220_A

; b. otherwise, set up for the right
;    set the left vs right channel flag indicating
;         that right channel in process
;    set the array register offsets to 32 sub-bands

_bita_210_A

;initialize for the possible presence of a sine wave in right
channel
;get the right xpsycho sine wave sub-bands to handle possible sine
wave

     move y:strtsinrgt,a       ;start entry equals 1st sub-band
     move a,y:strtsin          ;isolate the starting sub-band
     move y:endsinrgt,a        ;end entry equals last sub-band
     move a,y:endsin           ;isolate the ending sub-band

     bset #LEFT_vs_RIGHT,y:<stereo   ;flag for right channel in
progress
     move #NUMSUBBANDS,n0            ;offset to the right channel
SBMNRmax
     move n0,n1                ;offset to the right channel SBMsr
     move n0,n2                ;offset to right chan SBits
     move n0,n6                ;offset to right chan AtLimit
     move n0,n4                ;offset to right chan SBPos
     move n0,n5                ;offset to right chan SBIndx
```

```
_bita_220_A                    ;END of #NUMCHANNELS do loop

;set up for the initial allocation of the next subband

     move (r0)-                ;next sub-band SBMNRmax
     move (r1)-                ;next sub-band SBMsr
     move #16,r3               ;to position to next Allowed sb table
     move (r2)-                ;next sub-band SBits or JointSbits
     move (r3)-n3              ;next sub-band Allowed table array
     move r3,n3                ;set addr for next sub-band Allowed
pos
     move (r4)-                ;next sub-band SBPos
     move (r5)-                ;next sub-band SBIndx
     move y:count,r7           ;get current sub-band count
     move (r6)+                ;next sub-band AtLimit
     move r6,y:svereg          ;save updated AtLimit register
     move y:UsedSBReg,r6       ;get set to increment used counter
addr
     move (r7)+                ;increment the sub-band counter
     move (r6)+                ;next sub-band UsedSBs
     move r7,y:count           ;save new sub-band
     move r6,y:UsedSBReg       ;set incremented used counter addr
     move y:svereg,r6          ;restore AtLimit register

_bita_230_A                    ;END of #NUMSUBBANDS do loop

; if joint stereo does NOT apply, continue

     jclr #JOINT_FRAMING,y:<stereo,_bita_990_A

; if 2nd pass at Joint Stereo just completed, continue

     jset #JOINT_at_SB_BOUND,y:<stereo,_bita_990_A1

;if just finished the initial pass for JOINT stereo at FULL stereo
;    if frame could not handle full stereo, set up the joint

     jset #0,y:<jntflag,_bita_235_A
;    jclr #JOINT_at_FULL,y:<stereo,_bita_235_A

;the frame can handle FULL stereo, see if the previous frame
exhausted the
;  continuous joint boundary frame counter

     move y:jfrmcnt,a          ;frame decrement count at last
boundary
     move #>1,x0               ;to decrement frame count at last
bound
     sub  x0,a             ;decrement the joint frame counter
     move a,y:jfrmcnt          ;save new joint frame counter
     nop
     tst  a      #>FULL_STEREO,x1     ;see if frame count down over
                           ; & in case, set frame ISO stereo code
     jgt  <_bita_235_A         ;if joint, use last frame's sub-band
```


BAD ORIGINAL

```
cnt

;since the frame can handle FULL stereo, change the op frame type

     clr  a     x1,y:opfrtyp    ;to clear the history boundary
                               ; & set output frame as full stereo
     move a,y:boundist          ;clear   joint   history   sub-band
boundary
     move a,y:jfrmcnt           ;clear joint frame counter
     jmp  <_bita_990_A

_bita_235_A

;Joint at FULL stereo not possible, prepare for Joint Stereo
framing
;    store the demand rate

     bclr #JOINT_at_FULL,y:<stereo  ;clear flag FULL not possible
     move y:fixbits,x0          ;get the constant bit count
     move y:TotBits,a           ;get bits required for frame
     add  x0,a             ;set demand bits required
     move a,y:demand            ;save demand bit rate
     move y:frmtype,x1          ;output frame as joint stereo
     move x1,y:opfrtyp          ;set new output frame type=JOINT

;do the joint calculation routines and prepare the proper arrays

     move #>BOUND_4,x1          ;default to lowest boundary
     move x1,y:sibound          ;set sub-band boundary for jointval
     move #polydta,r0           ;addr of left channel poly samples
     move #polydta,r1           ;to set addr of right channel
     move #INPCM,n1        ;offset to right channel poly samples
     move #JntPlAnal,r2         ;joint channel poly samples
     move (r1)+n1               ;addr of right channel poly samples
     move #JntSBSKF,r3          ;addr of sub-band scale factors:
                           ;   the joint left and right scale
                           ;   factors
     move #JntSBMaxi,r4         ;joint channel Maxi factors by
                           ;   sub-band and block of 12 samples
     jsr  jointval         ;calculate joint array values

;set the intensity sub-band boundary

     move y:jntsub,a            ;get   sub-band   where   not   at   Mask
Thresh
     move y:q_psych,x0          ;get joint sub-band adjustment
     add  x0,a             ;adjust joint sub-band count

;based on some pre-determined minimum joint sub-band,
;  see if the sub-band count is to be forced to a higher value

     cmp  b,a              ;count vs pre-set minimum sub-band
     jge  <_bita_236_A          ;if count above minimum, continue
```


BAD ORIGINAL

```
;sub-band count is below the pre-determined joint sub-band

     move b,a               ;use pre-set minimum sub-band as count

_bita_236_A
     move #>BOUND_16,x1          ;start at highest boundary
     cmp  x1,a #>INTENSITY_16,y0    ;test limit vs sub-band
                           ; & get frame header boundary code
     jge  <_bita_240_A           ;we found the boundary
     move #>BOUND_12,x1          ;try the next highest boundary
     cmp  x1,a #>INTENSITY_12,y0    ;test limit vs sub-band
                           ; & get frame header boundary code
     jge  <_bita_240_A           ;we found the boundary
     move #>BOUND_8,x1           ;try the next highest boundary
     cmp  x1,a #>INTENSITY_8,y0     ;test limit vs sub-band
                           ; & get frame header boundary code
     jge  <_bita_240_A           ;we found the boundary
     move #>BOUND_4,x1           ;defaults to the lowest boundary
     move #>INTENSITY_4,y0       ;defaults to the lowest boundary

_bita_240_A

;test history of joint framing looking for a change in boundary

     mov  y:boundlst,a           ;get current boundary
     tst  a    y:jfrmcnt,b       ;see if set previously
                           ; & get frame decr counter
     jle  <_bita_242_AA          ;if not set, start new boundary &
count

;see if the frame decrement counter at zero

     tst  b                 ;see if zero (or less)
     jle  <_bita_242_AA          ;if done, start new boundary and
count

;compare last boundary to one just determined:
; if less, start with new higher boundary and restart the frame
decrement count
; if equal, continue without decrement frame counter
; else, decrement frame counter and switch to saved boundary and
ISO code

     cmp  x1,a y:jfrmcnt,r0      ;compare boundaries
                           ; & get curr decr frame count
     jlt  <_bita_242_AA          ;if less, start with new higher bound
     jeq  <_bita_248_AA          ;if equal, continue

;since new frame has boundary less that history boundary:
;    decrement frame counter
;    use history boundary
;    use history ISO code for the frame header

     move (r0)-                  ;decrement the frame counter
```


BAD ORIGINAL

```
     move y:boundlst,x1         ;switch to history intensity boundary
     move y:isocdelst,y0        ;switch to history boundary ISO code
     jmp  <_bita_248_AA

_bita_242_AA

;start new history at current frame's intensity boundary and
restart frame count

     move y:jntfrms,r0          ;initialize frame decrement count

_bita_248_AA

;set the frame header stereo intensity code

     move x1,y:sibound          ;set the sub-band boundary value
     move y0,y:stintns          ;for setsyst routine

;save current intensity boundary controls for the next frame

     move x1,y:boundlst         ;set last intensity boundary
     move y0,y:isocdelst        ;save ISO frame header code last used
     move r0,y:jfrmcnt          ;save frame decrement counter

;since doing joint stereo,
;pick correct joint scale factors for left channel then the right
channel
;first, see if testing with pickskf or pickjskf based on the factor
;applied to the demand bit rate with result compared to actual bit
rate

     bclr #1,y:<jntflag         ;use pickskf as default
     move y:p_psych,x1          ;get demand factor against demand
     move y:demand,x0           ;get the demand rate bit count
     mpy  x0,x1,a    y:AvlBits,x0   ;apply factor to demand bits
                     ; & get available bits

;if demand rate * factor gives a result still greater than the
actual bit rate,
; use pickskf because bits are at a premium, otherwise, use
pickjskf

     cmp  x0,a         ;see adjusted demand still higher
     jge  <_bita_242_A          ;if still higher, use pickskf
     bset #1,y:<jntflag         ;use pickjskf

_bita_242_A
     move #JntSBSKF,r0          ;addr of sub-band jnt scale
factors-left
     move #JntSBits,r1          ;addr of jnt SBits array-left channel
     jclr #1,y:<jntflag,_bita_243_A
;!!!dbg
;    nop
;    nop
```


BAD ORIGINAL

```
;     nop
;     nop
;     nop
;!!!dbg
      jsr  pickjskf          ;pick joint skf's for coding-left chan
      jmp  <_bita_244_A

_bita_243_A
;!!!dbg
;     nop
;     nop
;     nop
;     nop
;     nop
;!!!dbg
      jsr  pickskf           ;pick scale factors for coding-left chan
;!!!tst    jsr  pickjskf          ;pick joint skf's for coding-left
chan

_bita_244_A
      move #JntSBSKF,r0          ;addr   of   sub-band   jnt   scale
factors-left
      move #NUMSUBBANDS*NPERGROUP,n0      ;for  right  channel  SKFs
offset
      move #JntSBits,r1          ;addr of jnt SBits array-left channel
      move #NUMSUBBANDS,n1             ;for right channel SBits offset
      move (r0)+n0               ;adjust for the start of right chan
SKFs
      move (r1)+n1               ;adjust for start of right chan SBits

;see if testing with pickskf or pickjskf

      jclr #1,y:<jntflag,_bita_246_A
;!!!dbg
;     nop
;     nop
;     nop
;     nop
;     nop
;!!!dbg
      jsr  pickjskf          ;pick joint skf's for coding-rght chan
      jmp  <_bita_248_A

_bita_246_A
;!!!dbg
;     nop
;     nop
;     nop
;     nop
;     nop
;!!!dbg
      jsr  pickskf           ;pick scale factors for coding-rght chan
;!!!tst    jsr  pickjskf          ;pick joint skf's for coding-rght
chan
```


BAD ORIGINAL

```
_bita_248_A

;determine the joint stereo bits available for bit allocation

     bclr #JOINT_at_FULL,y:<stereo  ;now handle as joint stereo
     jsr  bitpool           ;set more available bits
     move x1,y:AvlBits

;restore original array of used sub-band count down counters for
joint allocate

     move #UsedSBs,r0
     move #SvUsedSBs,r1
     do   #NUMSUBBANDS*2,_bita_249_A
     move x:(r1)+,x0
     move x0,x:(r0)+

_bita_249_A

     jmp  <_bita_40_A          ;go   back   &   redo   the   initial
allocation

_bita_990_A

;if not joint stereo, store the demand rate

     move y:fixbits,x0         ;get the constant bit count
     move y:TotBits,a          ;get bits required for frame
     add  x0,a            ;set demand bits required
     move a,y:demand           ;save demand bit rate

_bita_990_A1

; done with the initial allocation phase, phase A
; set the de-allocation passes initial state of control flags

     bset #MASKING_PASS,y:<stereo       ;flag do masking passes
     bclr #HEARING_PASS,y:<stereo       ;allocate index must be >
1
     bclr #FINAL_PASS,y:<stereo         ;NOT final passes

;see if frame fits or do we have to de-allocate selectively

        move    y:TotBits,x0        ;get the total bits allocated
     move y:AvlBits,a          ;get available bits
     cmp  x0,a            ;TotBits vs BitsAvailable
     jge  <_bita_990_B         ;it fits, allocate any leftover bits

     do   #1000,_bita_990_B

;test the bit allocation timout flag
; if the timer flag was trip, switch over to the final bit
allocation
;    of any remaining bits
```

```
     jclr #0,y:<qtalloc,_bita_10_B
     jset #FINAL_PASS,y:<stereo,_bita_10_B    ;continue, if final
;    bset #FINAL_PASS,y:<stereo       ;set for FINAL criteria

;    ON_BITALLOC_LED_CD            ;tickle the led
     enddo                         ;stop the #1000 loop and exit
        move    y:TotBits,x0            ;get the total bits allocated
     jmp  <_bita_990_C             ;out of time, de-alloc under last
basis

_bita_10_B

;now let's look for qualifying candidates for next de-allocation
; we'll check out the left channel 1st, then the right

     bclr #LEFT_vs_RIGHT,y:<stereo    ;flag for left channel in
progress
     move #SBMNRmax,r0             ;addr of de-alloc Max signal-noise
        move    y:BInxAdd,r5            ;set register of SubBandIndex
array
        move    #AtLimit,r6             ;point to SubBandAtLimit array
     move #0,n0                    ;offset to the left channel SBMNRmax
     move n0,n5                    ;offset to left chan SBIndx
     move n0,n6                    ;offset to left chan AtLimit
     move #0,r2                    ;use r2 as a sub-band counter
     move r2,y:<MNRsub             ;start cnt of de-allocate table
entries
     move #>1,x1                   ;to test for index of 1
     move y:uselmsb,y1             ;to test for at least one alloc limit
     move #MNRval,n3               ;get address of MNRval table
     move #MNRsbc,n4               ;get address of MNRsbc table

;to deallocate the 1 index if the signal starts out below global
mask

     move #SBMsr,r1          ;addr of Mask-to-Signal by sub-band
     move n0,n1                    ;offset to left chan SBMsr
     move y:sibound,x0                   ;joint stereo intensity sub-band
     move x0,y:bandcnt                   ;bound subband decremented cntr
     bclr #JOINT_at_SB_BOUND,y:<stereo   ;clear reached boundary
sub-band

_bita_20_B

;loop thru the sub-bands for the current channel (left is 1st,
then, right)

     do   y:<usedsb,_bita_80_B

;to deallocate the 1 index if the signal starts out below global
mask

     jclr #JOINT_FRAMING,y:<stereo,_bita_21_B
     jset #JOINT_at_FULL,y:<stereo,_bita_21_B
```


BAD ORIGINAL

```
        jset #JOINT_at_SB_BOUND,y:<stereo,_bita_21_B
        move r3,y:svereg                     ;save reg 3
        move y:bandcnt,r3                    ;get decrement sub-band ctr
        jsr  chkjoint                 ;see if reached boundary
        move r3,y:bandcnt                    ;save new decremented ctr
        move y:svereg,r3                     ;restore the saved reg 3
        jclr #JOINT_at_SB_BOUND,y:<stereo,_bita_21_B
        move #NUMSUBBANDS*2,n1               ;Joint SBMsr values by sub-band

_bita_21_B

;if no index has been allocated, try the next sub-band

        move x:(r5+n5),a              ;check for an allocated index
        tst  a                  ;if zero, try the next sub-band
        jeq  <_bita_70_B              ;no allocation try next sub-band

;if a sine wave sub-band, do not deallocate

        jset #ALLOCATE_SINE,x:(r6+n6),_bita_70_B

;if the 3rd mode of selection, no checks are made

        jset #FINAL_PASS,y:<stereo,_bita_60_B   ;3rd mode, use this
one

;if 2nd mode of selection sub-band may be below the masking
threshold, but
;       checks to make sure that if index allocated is ONE and that
the
;       sub-band is not required for continity

        jset #HEARING_PASS,y:<stereo,_bita_50_B ;2nd mode num of index

;must be 1st mode of selection which requires that the sub-band
;     be below the masking threshold

        jclr #MASKING_LIMIT,x:(r6+n6),_bita_70_B       ;skip: above mask
thresh

_bita_50_B

;if we have allocated only 1 index, skip this sub-band if at least
one
;       allocation is required

        cmp  x1,a               ;see if index at 1
        jgt  <_bita_60_B               ;no, this sub-band qualifies

;to deallocate the 1 index if the signal starts out below global
mask

        move r2,a               ;get current sub-band
        cmp  y1,a               ;see if sub-band below at least 1
```


BAD ORIGINAL

```
     jge  <_bita_70_B            ;if greater, deallocation candidate
     move #>14,y1                ;if greater than 14, check
     cmp  y1,a y:uselmsb,y1      ;test sb vs 14
                                 ; & restore uselmsb to y1
     jlt  <_bita_70_B            ;if  less  than  14,  keep  the  1
allocation
     move x:(r1+n1),b            ;get Max Signal to MinMask
     tst  b                      ;if positive, started below global mask
     jle  <_bita_70_B            ;if  not  positive,  keep  the  1
allocation

_bita_60_B

;candidate qualifies,
; insert this candidate into the table for initial de-allocation

     jsr  insert_value

_bita_70_B

;advance to the next sub-band for the current channel

     move (r2)+                  ;increment the sub-band counter
     move (r0)+                  ;next sub-band SBMNRmax
     move (r5)+                  ;next sub-band SBIndx
     move (r6)+                  ;next sub-band AtLimit

_bita_80_B                       ;end of y:<usedsb do loop

;if we just finished the right channel,
;    let's see if we have any candidates to de-allocate

     jset #LEFT_vs_RIGHT,y:<stereo,_bita_90_B

;let's  go  thru  the  right  channel  looking  for  de-allocation
candidates

     bset #LEFT_vs_RIGHT,y:<stereo   ;flag  for  right  channel  in
progress
     move #SBMNRmax,r0           ;addr of de-alloc Max signal-noise
        move    y:BInxAdd,r5         ;set  register  of  SubBandIndex
array
        move     #AtLimit,r6         ;point to SubBandAtLimit array
     move #NUMSUBBANDS,n0            ;offset  to  the  right  channel
SBMNRmax
     move n0,n5                  ;offset to right chan SBIndx
     move n0,n6                  ;offset to right chan AtLimit
     move #0,r2                  ;use r2 as a sub-band counter
     move #MNRsbc,n4             ;get address of MNRsbc table

;to deallocate the 1 index if the signal starts out below global
mask

     move n0,n1                  ;offset to right chan SBMsr
```

```
     move y:sibound,x0          ;joint stereo intensity sub-band
     move x0,y:bandcnt          ;bound subband decremented cntr
     bclr #JOINT_at_SB_BOUND,y:<stereo   ;clear  reached  boundary
sub-band

     jmp  <_bita_20_B           ;no look thru right channel sub-bands

_bita_90_B

;if  there  are  any  entries  in  the  de-allocate  tables,  start
reclaiming bits

     move y:<MNRsub,a           ;get the de-allocate table entry cnt
     tst  a                 ;test for zero, no entries
     jne  <_bita_110_B          ;are  entries  at  this  criteria,
dealloc

;since there were no candidates to deallocate (MNRsub = 0),
; change the selection criteria:
;    if we've done the final criteria and nothing to de-allocate,
;         we can do nothing here, exit  (How Come???)
;    if we've not found anything with at least 2 indexes allocated,

;         switch to select from any sub-bands
;    if we've not found anything below the masking threshold,
;         switch to at least 2 indexes alloc
;redo the selection criteria

     jset #FINAL_PASS,y:<stereo,_bita_092_B
     jset #HEARING_PASS,y:<stereo,_bita_100_B
     jset #MASKING_PASS,y:<stereo,_bita_105_B
     bset #MASKING_PASS,y:<stereo
     jmp  <_bita_200_B          ;loop thru with this criteria

_bita_092_B

;see if a sine wave in either or both channels and if so open them
up for
;deallocation

     move #AtLimit,r6           ;address  of  AtLimit  array  both
channels
     jset #LEFT_SINE_WAVE,y:<stereo,_bita_94_B
     jset #RIGHT_SINE_WAVE,y:<stereo,_bita_96_B

;if no sine wave and still too much????? shouldn't be, exit

     enddo                      ;stop the #1000 loop and exit
       move    y:TotBits,x0        ;get the total bits allocated
     jmp  <_bita_990_C

_bita_94_B

;clear the sine wave indicators from left channel and open up for
```

```
dealloccation

     move y:strtsinlft,n6           ;1st  sine  wave  sub-band  of
adjacent pair
     bclr #LEFT_SINE_WAVE,y:<stereo      ;clear the indicator
     bclr #ALLOCATE_SINE,x:(r6+n6) ;clear the hold allocate if sine
     move y:endsinlft,n6       ;2nd sine wave sub-band of adjacent
pair
     nop
     bclr #ALLOCATE_SINE,x:(r6+n6) ;clear the hold allocate if sine
     jclr #RIGHT_SINE_WAVE,y:<stereo,_bita_200_B  ;loop with this
criteria

_bita_96_B

;clear the sine wave indicators from right channel and open up for
deallocation

     move #NUMSUBBANDS,n6           ;offset to right channel
     nop
     move (r6)+n6              ;shift over to right channel flags
     move y:strtsinrgt,n6           ;1st  sine  wave  sub-band  of
adjacent pair
     bclr #RIGHT_SINE_WAVE,y:<stereo     ;clear the indicator
     bclr #ALLOCATE_SINE,x:(r6+n6) ;clear the hold allocate if sine
     move y:endsinrgt,n6       ;2nd sine wave sub-band of adjacent
pair
     nop
     bclr #ALLOCATE_SINE,x:(r6+n6) ;clear the hold allocate if sine
     jmp  <_bita_200_B         ;loop thru with this criteria

_bita_100_B
     bclr #HEARING_PASS,y:<stereo
     bset #FINAL_PASS,y:<stereo
     jmp  <_bita_200_B         ;loop thru with this criteria

_bita_105_B
     bclr #MASKING_PASS,y:<stereo
     bset #HEARING_PASS,y:<stereo
     jmp  <_bita_200_B         ;loop thru with this criteria

;there are entries in the de-allocate tables

_bita_110_B

;de-allocate from the table from 1st entry to last
; or until enough bits have been reclaimed

     clr  a
     move a,y:count       ;start counter thru the table

;loop through the ordered de-allocation table

     do   y:<MNRsub,_bita_190_B
```


BAD ORIGINAL

```
     move #MNRsbc,n1               ;address of MNRsbc table
     move y:count,r0               ;current table entry index
     bclr #LEFT_vs_RIGHT,y:<stereo ;default to left channel
     bclr #JOINT_at_SB_BOUND,y:<stereo  ;clear  reached  boundary
sub-band
     move x:(r1+n0),a              ;get selected sub-band/channel
     move a,y:<MNRsb               ;isolate selected sub-band/channel
     move  (r0)+                   ;increment to next table entry
     move r0,y:count               ;save next table entry
     jclr #6,y:<MNRsb,_bita_120_B  ;if left channel, continue
     bclr #6,y:<MNRsb              ;right channel sub-band number
     move y:<MNRsb,a               ;get corrected sub-band (0-31)
     bset #LEFT_vs_RIGHT,y:<stereo ;indicate right channel

_bita_120_B

;restore the left channel array addresses

     move #SBMNRmax,r0            ;addr of de-alloc Max signal-noise
     move #SBMsr,r1        ;addr of Mask-to-Signal by sub-band
        move     y:BitsAdd,r2         ;set register of SBits array

;if doing a Joint Stereo frame (not upgraded to FULL),
; if the sub-band is included in the intensity coding,
;    set the SBMsr and SBits to the joint arrays

     jclr #JOINT_FRAMING,y:<stereo,_bita_130_B    ;NOT   Joint
framing
     jset #JOINT_at_FULL,y:<stereo,_bita_130_B    ;Joint upgraded
to FULL

;compare the selected sub-band to the stereo intensity sub-band
limit
;if not at or above the limit, continue as normal
;    otherwise switch to Joint array addresses

     move y:sibound,x0          ;get intensity sub-band limit
     cmp  x0,a             ;compare the two
     jlt  <_bita_130_B          ;we're doing a stereo sub-band

;we're at intensity sub-band limit
; shift over to Joint channel in SBMsr array (3rd set of sub-band
values in n1)
; and the Joint SBits

     bset #JOINT_at_SB_BOUND,y:<stereo  ;set   reached   boundary
sub-band
     move #NUMSUBBANDS*2,n1    ;Joint SBMsr values by sub-band
     nop
     move (r1)-n1               ;adjust addr to JointSBMsr
        move     #JntSBits,r2               ;set register of JointSBits
array

_bita_130_B
```

```
;continue restoring the left channel array addresses

        move    y:BPosAdd,r4        ;set register of SubBandPosition
array
        move    y:BInxAdd,r5        ;set register of SubBandIndex
array
        move    #AtLimit,r6         ;point to SubBandAtLimit array

;if from left channel, addresses are OK. otherwise, offset for
right channel

     jclr #LEFT_vs_RIGHT,y:<stereo,_bita_140_B

     move #NUMSUBBANDS,n0           ;offset to the right channel
SBMNRmax
     move n0,n1                ;offset to the right channel SBMsr
     move n0,n2                ;offset to right chan SBits
     move n0,n4                ;offset to right chan SBPos
     move n0,n5                ;offset to right chan SBIndx
     move n0,n6                ;offset to right chan AtLimit
     move (r0)+n0              ;offset register for SBMNRmax to
right
     move (r1)+n1              ;offset register for SBMsr to right
     move (r2)+n2              ;offset register for SBBits to right
     move (r4)+n4              ;offset register for SBPos to right
     move (r5)+n5              ;offset register for SBIndx to right
     move (r6)+n6              ;offset register for AtLimit to right

_bita_140_B

;set the proper allowed table of indexed position based on the
selected sub-band

        move    y:AllwAdd,r3        ;init the current Allowed table
     tst  a               ;see if it's sub-band zero (from above)
     jeq  <_bita_150_B         ;sub-band zero was selected
     move #16,n3               ;to increment to next sub-band addr
     do   a,_bita_150_B        ;increment to sub-band number chosen
     move (r3)+n3              ;16 position entries per sub-band

_bita_150_B
     move r3,n3                ;set Allowed addr for sub-band chosen
     move y:<MNRsb,n0          ;get selected sub-band in SBMNRmax
     move n0,n1                ;sub-band in SBMsr
     move n0,n2                ;sub-band in SBits
     move n0,n4                ;sub-band in SBPos
     move n0,n5                ;sub-band in SBIndx
     move n0,n6                ;sub-band in AtLimit
     move y:ndatabit,r7        ;addr of NDataBit count by position
     move y:TotBits,a          ;get current bits allocated
     move x:(r5+n5),r3         ;get the current allocated index
     move x:(r4+n4),n7         ;get the position at the old index
     move (r3)-                ;back up one index
     move r3,x:(r5+n5)         ;save new SBIndx for sub-band
```

```
     move y:(r7+n7),x0          ;data bits allocated at that position
     sub  x0,a               ;subtract old allocated data bits
     move x:(r3+n3),n7          ;get new position
     move n7,x:(r4+n4)          ;save new SBPos for sub-band
     move y:(r7+n7),b           ;data bits allocated at new position
     add  b,a                ;add new allocated data bits

     tst  b                  ;see if index 1 just de-allocated
     jne  <_bita_160_B          ;if not, save the new TotBits value

;we have to take off the scale factor bits

     move x:(r2+n2),n7          ;get  the  SBits  scale  factor  code
(0-3)
     move #NSKFBits,r7          ;addr SBits scale factor bit count
tbl
     nop
     move y:(r7+n7),y0          ;get the scale factor bit count
     sub  y0,a               ;subtract from TotBits

;if  joint  stereo and  we  have  reached  the  intensity  sub-band
boundary
;    add the right channel joint SBits bit count also for this
sub-band

     jclr #JOINT_at_SB_BOUND,y:<stereo,_bita_160_B
     move #>NUMSUBBANDS,b             ;offset to right channel Joint
SBits
     move n2,x0                 ;sub-band
     add  x0,b               ;offset to right channel subband
     move b1,n2                 ;access  right  channel-sband  Joint
SBits
     nop
     move x:(r2+n2),n7          ;get  the  SBits  scale  factor  code
(0-3)
     nop
     move y:(r7+n7),y0          ;save the scale factor bit count
     sub  y0,a               ;subtract from TotBits

_bita_160_B
     move a,y:TotBits           ;save the new total bits

;check if Signal-to-Noise position that Signal above/below Masking
Threshold

     bclr #MASKING_LIMIT,x:(r6+n6)   ;clear AtLimit below masking
threshold
     move x:(r4+n4),n7          ;get the position
     move #SNR,r7               ;addr of Signal-to-Noise table
     move x:(r1+n1),y0          ;get signal to mask ratio
     move y:(r7+n7),a           ;get the Signal-Noise at position
     add  y0,a x:(r5+n5),r3     ;add MNR to SNR for test
                                ; & set up to set prev index for its pos
     jle  <_bita_170_B          ;above mask, skip next statement
```

```
    bset #MASKING_LIMIT,x: r6-n6     ;set AtLimit below masking
threshold

_bita_170_B

;check if the bit pool can now handle the frame as allocated

    move y:TotBits,a            ;get the new total bits
    move y:AvlBits,x0           ;get the available bits
    cmp  x0,a                 ;BitsAvailable vs TotBits
    jgt  <_bita_180_B           ;need    more,    continue    with
de-allocation

    enddo                       ;we're done here, stop MNRsub loop
    enddo                       ;we're done here, stop #1000 loop
    jmp  <_bita_990_B

_bita_180_B

;if there is no index allocated (r3 = 0), continue with the next
table entry

    move r3,a                 ;get newly decremented index allocated
    tst  a     (r3)-            ;if it is zero, continue
                              ; & back up one index for that position
    jeq  <_bita_185_B           ;allocated index equals 0, continue

;set the value for testing the best sub-band to deallocate bits
from
;if the frame cannot handle the full required allocation

    move x:(r3+n3),n7           ;get the position at the previous
index
    nop
    move y:(r7+n7),a            ;get the Signal-Noise at position
    add  y0,a                 ;calc Sig-to-Noise at prev position
    move a,x:(r0+n0)            ;save in SBMNRmax array for later

_bita_185_B                   ;continue y:<MNRsub do loop
    nop

_bita_190_B                     ;end of y:<MNRsub do loop
    nop

_bita_200_B                   ;continue #1000 do loop
    nop

_bita_990_B                     ;end of #1000 do loop

; set the allocation passes initial state of control flags

    bset #MASKING_PASS,y:<stereo        ;flag do masking passes
    bclr #HEARING_PASS,y:<stereo        ;NOT   hearing   threshold
passes
```


BAD ORIGINAL

```
     bclr #FINAL_PASS,y:<stereo          ;NOT final passes

;get the total bits allocated so far

        move    y:TotBits,x0

; Now that we have the initial bit allocation, iterate on it.
;
;(c  for  LoopCount = 1; ; --LoopCount   {
;
        do      #1000,_bita_990_C

;test the bit allocation timout flag
; if the timer flag was trip, switch over to the final bit
allocation
;    of any remaining bits

     jclr #0,y:<qtalloc,_bita_10_C
     jset #FINAL_PASS,y:<stereo,_bita_10_C
     bset #FINAL_PASS,y:<stereo

;    ON_BITALLOC_LED_CD       ;tickle the led

;this is equivalent to the call to the c subroutine:
;
;(c) AllocateBits()
;
;inititial allocation is done, set-up for as needed allocation loop
;restore the left channel array addresses

_bita_10_C
        move    #SBMsr,r1           ;set register of SBMsr array
        move    y:BitsAdd,r2        ;set register of SBits array
        move    y:BPosAdd,r4        ;set register of SubBandPosition
array
        move    y:BInxAdd,r5        ;set register of SubBandIndex
array
        move    #AtLimit,r6         ;point to SubBandAtLimit array
;
;(c)       FirstTime = 1;      /*start run thru subbands this time
*/
;
     bset #FIRST_TIME,y:<stereo    ;FirstTime = !0

     clr  a
     move a1,y:count            ;start the sub-band counter
        move    y:AllwAdd,r0        ;init the current Allowed table
        move    #SNR,r3

;in case of joint stereo, clear the reached intensity sub-band
boundary flag

     move y:sibound,x1          ;joint stereo intensity sub-band
     move x1,y:bandcnt          ;bound subband decremented cntr
```

```
    bclr #JOINT_at_SB_BOUND,y:<stereo   ;clear  reached  boundary
sub-band

;go through all used sub-bands for both channels looking at only
those
; that have not reached the allocation limit

        do      y:<usedsb,_bita_130_C

;clear the n registers for the left channel reference

    clr  a
    move a,n1              ;SBMsr array
    move a,n2              ;SBits array
    move a,n4              ;SBPos array
    move a,n5              ;SBIndx array
    move a,n6              ;AtLimit array

;clear the left vs right channel flag indicating that left channel
in process

    bclr #LEFT_vs_RIGHT,y:<stereo   ;flag  for  left  channel  in
progress

; if joint stereo does NOT apply, continue

    jclr #JOINT_FRAMING,y:<stereo,_bita_30_C

; if joint stereo upgraded to full, continue

    jset #JOINT_at_FULL,y:<stereo,_bita_30_C

; if doing joint stereo and have already switched over to joint
SBits array,
;    but now have to adjust to 3rd set of SBMsr values

    jset #JOINT_at_SB_BOUND,y:<stereo,_bita_20_C

; see if the joint stereo intensity sub-boundary has been reached
; if not, continue at full stereo for these early sub-bands
; otherwise, switch over to the JointSBits and Joint channel in
;    the SBMsr array (3rd set of sub-band values (n1

    move r3,y:svereg           ;save reg 3
    move y:bandcnt,r3          ;get decrement sub-band ctr
    jsr  chkjoint        ;see if reached boundary
    move r3,y:bandcnt          ;save new decremented ctr
    move y:svereg,r3           ;restore the saved reg 3
    jclr #JOINT_at_SB_BOUND,y:<stereo,_bita_30_C

    move #JntSBits,r2          ;shift over to Joint SBits array
    move y:count,n2            ;to offset to current sub-band
    nop
    move (r2)+n2               ;adj addr to current sub-band
```

```
     move #1,n1                      ;reset to left channel

_bita_20_C

;we're at intensity sub-band limit
; shift over to Joint channel in SBMsr array
;      3rd set of sub-band values in n1

     move #NUMSUBBANDS*2,n1          ;Joint SBMsr values by sub-band

_bita_30_C

;process the current channel

     do    #NUMCHANNELS,_bita_120_C

;see if this sub-band's limit flag was set previously, and skip if
it has
;
;(c)        if( Left(or Right)AtLimit[SubBand] )
;(c)             continue;
;
        jset    #ALLOCATE_LIMIT,x:(r6+n6),_bita_100_C ;skip subbnd
reached limit

     jset #FINAL_PASS,y:<stereo,_bita_40_C  ;pass skips below mask
check

        jset    #MASKING_LIMIT,x:(r6+n6),_bita_100_C ;skip subband
reached limit

_bita_40_C
        move     x:(r4+n4),a         ;get curr position[SubBand]

;see if this sub-band has reached its limit already
;
;(c)        if( Left(or Right)SubBandPosition[SubBand] == MaxPos ) {
;(c)             Left(or Right)AtLimit = 1;
;(c)             continue;
;(c)        }
;
     move y:MaxPos,y0            ;set max position
        cmp      y0,a     a1,n3           ;see if max position
                       ;  & move pos to n3
        jeq      _bita_80_C          ;reached its allocation limit,
set flag

;check this sub-band out
;  see if there is room to handle the next allocation for this
sub-band
;
;(c)        NextSubBandPosition =
;(c)             AllowedPositions[SubBand]
;(c)                  [Left(orRight)SubBandIndex[SubBand]+1];
```


BAD ORIGINAL

```
; c              TestBits = OldTotBits
; c                - NDataBits[Left orRight SubBandPosition[SubBand]]
; c                - NDataBits[NextSubBandPosition];
; c          if  Left:or Right:SubBandIndex[SubBand] == 0
; c                TestBits -= NSKFBitsOLeft:or Right:SBits[SubBand]];
;
     clr  b     #>1,y1           ;init added scale factor bits
                          ; & to incr to next allowed bits size
        move    x:(r5+n5),a         ;SubBandIndex[SubBand]

;if this will be the 1st index, we must account for the scale
factor bits

     tst  a     #NSKFBits,r7    ;see if 0
                          ; & set addr of NSKFBits array
     jne  <_bita_50_C           ;not 1st index, skip add scale bits

;set the scale factor - sbits needed for this 1st index in this
sub-band

     move x:(r2+n2),n7          ;get SBIts index
     nop
     move y:(r7+n7),b           ;num bits for scaling info

;if joint stereo and we have reached the intensity sub-band
boundary
;    add the right channel joint SBits bit count also

     jclr #JOINT_at_SB_BOUND,y:<stereo,_bita_50_C
     move #NUMSUBBANDS,n2                 ;offset to right channel Joint
SBits
     nop
     move x:(r2+n2),n7          ;get the SBits scale factor code
(0-3)
     move #0,n2                 ;restore to left channel Joint SBits
     move y:(r7+n7),y0          ;save the scale factor bit count
     add  y0,b            ;add left to right Joint SBits cnt

_bita_50_C
     add  y1,a y:ndatabit,r7  ;increment
                          ; & addr of NDataBit count by position
     move a1,n0                 ;set offset for Allowed next index

;do not allocate position 17, stop this sub-band if that is the
case

     move #>17,y1               ;get the biggest position to test
     move x:(r0+n0),a           ;get the NextPosition as the new pos
     cmp  y1,a            ;see if biggest allocation
     jeq  <_bita_80_C           ;do not, end allocation this sub-band

;see if next allocation is passed the max for this sub-band as per
Allowed table
;  this has happened if this next position is zero
```

```
     tst  a    a1,n7            ;see if passed the maximum position
                            ; & move new pos to n7
     jeq     <_bita_80_C              ;reached its allocation limit,
set flag

;test the allocation at this new position

     move y::r7+n7),y1          ;get NDataBits[NextSBPos]
     add  y1,b n3,n7            ;add to any scaling info bits
                            ; & set offset SubBandPos[SubBand]]
     move b1,y1                 ;bits to add for next index
     move x0,b              ;b==>TestBits = OldTotBits
     move y:(r7+n7),y0          ;get NDataBits[SBPos[SubBand]]

     sub  y0,b a1,x1            ;TestBits -= current bits
                            ; & put new position in proper reg
     add  y1,b y:AvlBits,a      ; TestBits += next allocation bits
                            ;  & gets BitsAvaliable

;
;(c)        if( TestBits > BitsAvailable ) {
;(c)             Left(or Right)AtLimit = 1;
;(c)             continue;
;(c)        }
;
     cmp  b,a  b,y:TotBits      ;see if room & save allocation
     jlt  <_bita_80_C           ;  no  room,  set  as  AtLimit  and
continue

;if this is the final loop, skip the next test and allocate the
bits

     jset #FINAL_PASS,y:<stereo,_bita_70_C  ;pass skips below mask
check

;
;(c)        SMR = Left(or Right)SubBandMax[SubBand]
;(c)                - Left(or Right)MinMaskingDb[SubBand]
;(c)        MNR = SNR[Left(or Right)SubBandPosition[SubBand]] - SMR
;
     move y:(r3+n3),y1          ;get SNR[SubBandPos[SubBand]]
        move     x:(r1+n1),a        ;SBMsr[SubBand] Mask-to-Signal
        add      y1,a      y:MNRmin,b    ;add Sig-Noise ratio;
                            ; & get MNRmin for below
     jgt  <_bita_90_C           ;below  Masking,  go  to  take  out
partially
;
;(c)        if( FirstTime || MNR < MNRmin ) {
;(c)             MNRsb = SubBand;
;(c)             MNRchan = channel;
;(c)             MNRMin = MNR;
;(c)             FirstTime = 0;
;(c)        }
;
        move     a,y1                   ;save MNR
```


BAD ORIGINAL

```
        jset    #FIRST_TIME,y:<stereo,_bita_60_C ;if first, save as
minimum
        cmp     y1,b                            ;MNRmin - MNR
        jle     _bita_100_C

_bita_60_C
     move n0,y:MNRinx           ;MNRinx = NewIndex;
     move x1,y:MNRpos           ;MNRpos = NewPosition;
     move y:TotBits,x1          ;get the allocation of bits
     move x1,y:HldBits          ;save the allocation of bits
     move y:count,x1            ;get current sub-band
        move    x1,y:<MNRsb          ;MNRsb = SubBand;
        move    n2,y:MNRchan         ;MNRchan = 0 if left,32 if right
     move y1,y:MNRmin           ;MNRmin = MNR;
     bclr #FIRST_TIME,y:<stereo     ;clear FirstTime flag

        jmp     _bita_100_C

;we are on the final allocations passes after all channels for all
sub-bands
;    are driven below the Global Masking threshold

_bita_70_C
     move y:TotBits,x0          ;save new TotBits
     move n0,x:(r5+n5)          ;save new sub-band index
     move x1,x:(r4+n4)          ;save new allocation position
     bclr #FIRST_TIME,y:<stereo     ;clear FirstTime flag
     jmp  <_bita_100_C

_bita_80_C
        bset      #ALLOCATE_LIMIT,x:(r6+n6)  ;set  the  completely
allocated bit
        bset      #HEARING_LIMIT,x:(r6+n6)   ;set  the  completely
allocated bit

_bita_90_C
        bset     #MASKING_LIMIT,x:(r6+n6)  ;set the reached global
masking bit

;finished the sub-band at the current channel,
; a. if just finished the right, skip next instructions

_bita_100_C
      jclr #LEFT_vs_RIGHT,y:<stereo,_bita_110_C
      enddo
      jmp  <_bita_120_C

; b. otherwise, set up for the right
;     set the left vs right channel flag indicating
;          that right channel in process
;     set the array register offsets to 32 sub-bands

_bita_110_C
      bset #LEFT_vs_RIGHT,y:<stereo   ;flag for right channel in
```

```
progress
    move #NUMSUBBANDS,n1          ;offset to the right channel
SBMsr
    move n1,n2                    ;offset to right chan SBits
    move n1,n6                    ;offset to right chan AtLimit
    move n1,n4                    ;offset to right chan SBPos
    move n1,n5                    ;offset to right chan SBIndx

;finished both channels for this sub-band, now set up for the next
subband

_bita_120_C
    move y:count,r7               ;get current sub-band to increment
       move    #16,n0                ;now update Allowed to next
sub_band
    move (r1)+                    ;SBMsr array
    move (r2)+                    ;SBits array
    move (r4)+                    ;SBPos array
    move (r5)+                    ;SBIndx array
    move (r6)+                    ;AtLimit array
       move    (r0)+n0               ;advance  Allowed  to  next
sub-band
    move (r7)+                    ;increment the sub-band counter
    move r7,y:count               ;save new sub-band number

_bita_130_C

; At this point the following registers are in use
;       y:AvlBits = # of bits available
;       y:<MNRsb = MNRsb
;       y:MNRMin = MNRmin

;We test now to see if this trip thru the loop produced any changes
; and if not, we have finished the bit allocation for this frame.
;
;(c) if( FirstTime )
;(c)      return;
;
        jclr   #FIRST_TIME,y:<stereo,_bita_140_C  ;not 1st, alloc
to selected
        jclr   #FINAL_PASS,y:<stereo,_bita_160_C  ;not final, set
1 more loop

;finished, end the loop and go to exit routine

        enddo
     jmp  <_bita_990_C

_bita_140_C

;test flag all candidates are below masking threshold

        jset   #FINAL_PASS,y:<stereo,_bita_170_C    ;if  final,
allocated already
```


BAD ORIGINAL

```
;restore the left channel array addresses

        move    y:BPosAdd,r4          ;set register of SubBandPosition
array
        move    y:BInxAdd,r5          ;set  register  of  SubBandIndex
array
     move y:MNRchan,a             ;get indication as to which channel
     tst  a                  ;0 if from left channel
     jeq  <_bita_150_C            ;if from left channel, addrs OK
     move #NUMSUBBANDS,n4              ;offset to right SBPos channels
     move n4,n5                   ;offset to right SBIndx channels
     move (r4)+n4                 ;offset register for SBPos to right
     move (r5)+n5                 ;offset register for SBIndx to right

_bita_150_C

;       SubBandIndex[MNRsb]++
;                              SubBandPosition[MNRsb]     =
AllowedPositions[MNRsb][SubBandIndex[MNRsb]]

        move    y:<MNRsb,n5           ;MNRsb
        move    n5,n4                      ;MNRsb
     move y:MNRinx,x1             ;get the saved new index
     move x1,x:(r5+n5)            ;update  the  SBIndx  for  selected
sub-band
     move y:MNRpos,x1             ;get the saved new Allowed position
     move x1,x:(r4+n4)            ;update  the  SBPos  for  selected
sub-band
     move y:HldBits,x0            ;set the new bit allocation total cnt
     jmp  <_bita_170_C            ;continue major loop

;now lets just allocate what's left now that all are below mask

_bita_160_C
     bset #FINAL_PASS,y:<stereo    ; just loop now

_bita_170_C
        nop

_bita_990_C

; restore the register 7 values

     move x:bitallocM7Save,m7
     move x:bitallocN7Save,n7
     move x:bitallocR7Save,r7

;    bclr WATCH_DOG        ;tickle the dog
;    nop

;check for any sub-band with no index allocation
;  and if zero, zero it's in use count down counter

     move y:BInxAdd,r5          ;addr allocated indexes
```

```
     move #UsedSBs,r6          ;addr of used sub-band counters
     do   #NUMSUBBANDS*2,_bita_994_C

     move x:(r5)+,a       ;get allocated index
     tst  a               ;see if zero
     jne  <_bita_992_C         ;if not zero, continue
     move a,x:(r6)        ;reset count down counter to zero

_bita_992_C
     move (r6)+                ;incr addr to next counter

_bita_994_C

;set up the padded bits count for ancillary data

     move x0,y:TotBits         ;save bits actually allocated
     move y:AvlBits,b          ;determine number of bits padded
     sub  x0,b            ;bits available minus total allocated
     move b1,y:padbits         ;save count of unallocated audio bits
        rts

;insert_value():
;
;This routine orders the table of values per sub-band/channel
;that are to be de-allocated as needed. The table is ordered in
;descending sequence that makes the 1st entry the one that can best
;afford a deallocation.

;on entry:
;
;    x:(r0+n0) = the current value to be inserted
;    y:<stereo = bit 1 indicates left channel (0) or right channel
(1)
;    r2 = the sub-band number to be inserted
;    y:<MNRsub  =  current  count  of  entries  in  the  ordered
deallocation tables
;    n3 = address of MNRval table
;    n4 = address of MNRsbc table
;
;on exit:
;
;    y:<MNRsub  =  incremented  count  of  entries  in  ordered
deallocation tables
;
;    a = destroyed
;    b = destroyed
;    x0 = destroyed
;    y0 = destroyed
;    r3 = destroyed
;    r4 = destroyed

     org  phe:
;    org  pli:
```


BAD ORIGINAL

```
insert_value

;get the current value to be inserted and set upo the start into
; the ordered table of values and the assoicated table of
sub-band/channels

     move x:(r3+n0),a          ;get the current value to insert
     move y:<MNRsub,b          ;get current count of table entries

;if this is the 1st value to be inserted ino the table, skip the
; search for its place and enter this as table entry no 1

     tst  b    #0,r3           ;see if this is 1st entry into table
                         ; & set to 1st entry in MNRval table
     jeq  <_insert_50          ;if 1st, skip following table search

;search through the table of entries so far established looking for
where
;to store this current value

     do   y:<MNRsub,_insert_20

     move x:(r3+n3),x0         ;get the table value for comparison
     cmp  x0,a           ;against the new value to be inserted
     jlt  <_insert_10          ;if less, value is further down table

;when the new value is greater than or equal to the table entry,
; this is its place in the table, we may have to shift the
following
; table entries in order to enter this new value

     enddo                     ;stop the y:<MNRsub do loop
     jmp  <_insert_20          ;see if the table must be shifted

_insert_10
     move (r3)+                ;try the next table entry

_insert_20                     ;end of y:<MNRsub do loop

;if this entry number (its place in the table) equals the count of
entries,
; this entry will be the new LAST entry in the table

     move r3,x0                ;get its place in the table to
compare
     cmp  x0,b           ;its place to current table entry count
     jgt  <_insert_25          ;if less, we have to shift the table
     jeq  <_insert_50          ;if eq, entry is appended to the
table
     move b1,r3                ;?? let's make sure we use last entry
     jmp  <_insert_50

_insert_25
```


BAD ORIGINAL

```
;we need to shift the subsequent entries in the table down one and
then
; insert this new sub-band/channel value

     move b1,r3                 ;establish the curr table ends
     move b1,r4                 ;for both MNRval and MNRsbc
     move (r3)+n3               ;set r3 with addr of MNRval end - 1
     move (r4)+n4               ;set r4 with addr of MNRsbc end + 1
     move (r3)-                 ;back off 1 to get last MNRval entry
     sub  x0,b  (r4)-           ;number of table entries to shift
                              ; & back off 1 to get last MNRsbc entry

     do   b,_insert_40          ;shift each down 1 position in tables

     move x:(r3)+,y0            ;get curr value and incr to rec addr
     move y0,x:(r3)-            ;put value 1 entry down & back up 1
     move x:(r4)+,y0            ;curr sub-band/chan & incr to rec
addr
     move y0,x:(r4)-            ;put value 1 entry down & back up 1
     move (r3)-                 ;back up one more entry table MNRval
     move (r4)-                 ;back up one more entry table MNRsbc

_insert_40                      ; end of b do loop

;restore entry location to receive value and sub-band/channel

     move x0,r3

_insert_50

;insert the current value at this location in the ordered table
; also insert the sub-band number and set the channel flag

     move r3,r4                 ;matching position in the MNRsbc
table
     move a,x:(r3+n3)           ;enter sorted value
     move r2,x:(r4+n4)          ;enter the sub-band number
     jclr #LEFT_vs_RIGHT,y:<stereo,_insert_99
     bset #6,x:(r4+n4)          ;flag as the right channel

_insert_99

;increment the count of entries in the ordered deallocation tables

     move y:<MNRsub,r3          ;we need to increment entry counter
     nop
     move (r3)+
     move r3,y:<MNRsub          ;save the new table entry count

     rts
```

```
        opt     fc,mex
;
;  c  1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
   \XCODE\xcode.asm: the ULTIMA cdq2000 with 2 xpsycho's and xcode in one DSP

;updated to support 48000, 44100, 32000, 24000, 22050 and 16000 sampling rates

        title   'MUSICAM Transitter Main'
        stitle  'Initialization'

        include 'def.asm'
        include '..\common\ioequ.asm'
        include 'box_ctl.asm'
        include 'box_tbls.asm'
        include 'translte.asm'

        page

;In a given MUSICAM frame time period this routine performs the XPSYCHO
;function on both channels followed by the XCODE functions of bit
;allocation and frame encoding.

        section lowmisc
        xdef    word_out
        xdef    word_in1
        xdef    word_in2
        xdef    word_in3
        xdef    not_appl
        xdef    starty
        xdef    maxsubs
        xdef    maxcritbnds
        xdef    ipwptr
        xdef    frmbits
        xdef    fixbits
        xdef    audbits
        xdef    usedsb
        xdef    stereo
        xdef    cmprsctl
        xdef    sibound
        xdef    nmskfreqs
        xdef    cutmus
        xdef    outsize
        xdef    timer
        xdef    timeout
        xdef    qtalloc
        xdef    oprptr
        xdef    frmstrt
        xdef    frmnext
        xdef    plctmn
        xdef    plctl1
        xdef    plctl2
        xdef    dbgcnt
        xdef    endy

        xdef    limitsb

        org     yli:
stxcode_yli
```


BAD ORIGINAL

```
word_out        ds      1       ;applicable output (leds, switches)
word_in1        ds      1       ;applicable input (switches, lines)
word_in2        ds      1       ;applicable input (switches, lines)
word_in3        ds      1       ;applicable input (switches, lines)
  _appl         ds      1       ;satisfy non-applicable hardware settings

starty

maxsubs         ds      1       ;working MAXSUBBANDS for sample/bit rates
maxcritbnds     ds      1       ;MAXCRITBNDS for sample/bit rates
ipwptr          ds      1       ;input PCM buffer write pointer (even = left)
frmbits         ds      1       ;bits in the audio portion of frame
fixbits         ds      1       ;bits required before audio data bits
audbits         ds      1       ;number of bits available for audio data
usedsb          ds      1       ;number of used sub-bands
stereo          ds      1       ;y:<stereo = flags:
                                ;bit 0 means stereo vs mono framing
                                ;  0 = stereo framing
                                ;  1 = mono framing
                                ;bit 1 indicates left vs right channel
                                ;  0 = looping thru left channel arrays
                                ;  1 = looping thru right channel arrays
                                ;bit 2 indicates joint stereo applies
                                ;  0 = NOT joint stereo framing type
                                ;  1 = IS joint stereo framing type
                                ;bit 3 indicates curr frame upgraded to
                                ; full stereo by joint bit allocation
                                ;        (if joint stereo applies)
                                ;  0 = normal joint stereo allocation
                                ;  1 = FULL STEREO allocation
                                ;bit 4 indicates the stereo intensity
                                ; sub-band boundary has been reached
                                ;        (if joint stereo applies)
                                ;  0 = NO sub-bands still below
                                ;        intensity boundary
                                ;  1 = sub-bands above intensity
                                ;        boundary
                                ;bit 5 is FirstTime switch in a loop
                                ; thru the bit allocation
                                ;  0 = cleared if any allocations
                                ;        were made
                                ;  1 = no allocations made to any
                                ;        sub-band
                                ;bit 6 indicates a below masking
                                ;        threshold allocation pass
                                ;  0 = some sub-bands not below mask
                                ;  1 = all sub-bands are below mask
                                ;bit 7 indicates a below hearing
                                ;        threshold allocation pass
                                ;  0 = some sub-bands not below hearing
                                ;        threshold
                                ;  1 = all sub-bands are below hearing
                                ;        threshold
                                ;bit 8 indicates final bit allocation
                                ; passes to use up any available bits
                                ;  0 = not yet
                                ;  1 = allocate remainder in bit pool
                                ;bit 9 indicates limit of sub-bands requiring
                                ; at least one position has been reached:
                                ;  0 = not yet, 1 = limit reached
```

```
                                        ;bit 10 indicates maximum limit of sub-bands
                                        ; that are to be allocated has been reached:
                                        ;  0 = not yet, 1 = limit reached
                                        ;bit 11 indicates whether or not dual
                                        ; transmission output lines apply and
                                        ; that the block sequence number must be
                                        ; appended to the frame
                                        ;  0 = NO block sequence number
                                        ;  1 = YES append the block sequence number
                                        ;bit 12 indicates that the split framing mode
                                        ; applies (go to MONO if one line is down)
                                        ;  0 = split framing does not apply
                                        ;  1 = split framing does apply
                                        ;bit 13 indicates to do a split mono frame
                                        ; because one line is down
                                        ;  0 = code a normal frame
                                        ;  1 = code a split mono frame
                                        ;
cmprsctl        ds      1               ;control flags for CCS compression:
                                        ;  bit 0 = application:
                                        ;       0 = ISO standard
                                        ;       1 = CCS compression applies
sibound         ds      1               ;intensity subband boundary alloc
nmskfreqs       ds      1               ;NMSKFREQS for sample/bit rates
outmus          ds      1               ;number of words to read in
outsize         ds      1               ;circular buffer ctl frame o/p buffer
timer           dc      0               ;frame sync timer interrupt sensor:
                                        ;  bit 0 set by irqb - received frame sync
                                        ;  bit 1 after 1st frame skipped if sync failure
  meout         dc      0               ;frame sync failure counter
 .alloc         ds      1               ;frame msec timer interrupt bit alloc
                                        ; signal bit allocator to finish up
oprptr          ds      1               ;read pointer into frame buffer
frmstrt         ds      1               ;starting addr of current frame
frmnext         ds      1               ;start addr of frame 2 to align with frame sync
plctmn          ds      1               ;successive phase lock detect high conter main
plctl1          ds      1               ;successive phase lock detect high conter line 1
plctl2          ds      1               ;successive phase lock detect high conter line 2
dbgcnt          ds      1             ;!!! debug counter

endy

limitsb dc      0       ;LIMITSUBBANDS  ;sub-bands req at least 1 allocation

endxcode_yli
        endsec

        section lowmisc
        xdef    startx_xli
        xdef    polyst
        xdef    ntonals
        xdef    nmasker
        xdef    nalislft
        xdef    nalisrgt
        xdef    maxtonal
        xdef    maxbin
        xdef    sinbin
        xdef    sincnt
        xdef    sintest
        xdef    SvReg0
```


BAD ORIGINAL

```
        xdef    dbptr
        xdef    endx_xli

        org     xli:
  strtx_xli
polyst          ds      1       ;addr of the polyanalysis start
ntonals         ds      1       ;number of tonals in tonal structure
nmasker         ds      1       ;number of maskers in masker structure
nalislft        ds      1       ;number aliasers - left channel
nalisrgt        ds      1       ;number aliasers - right channel
maxtonal        ds      1       ;to see if sine wave, highest tonal
maxbin          ds      1       ;if sine wave, bin num of highest tonal
sinbin          ds      1       ;bin number of sine wave must persist
sincnt          ds      1       ;frame cnt to see if sine wave persists
sintest         ds      1       ;channel tester to see if sine wave
SvReg0          ds      1       ;Save Register 0
dbptr           ds      1     ;!!!debug
endx_xli
        endsec

        section highmisc

        xdef    bitrate
        xdef    frmrate
        xdef    rawrate
        xdef    smplrte
        xdef    smplcde
        xdef    smplidbit
        xdef    padrate
        xdef    paddiff
        xdef    padrest
        xdef    usediff
        xdef    bndwdth
        xdef    frmtype
        xdef    opfrtyp
        xdef    maxsubbands
        xdef    stintns
        xdef    oldccs
        xdef    strtsinlft
        xdef    endsinlft
        xdef    strtsinrgt
        xdef    endsinrgt
        xdef    sincntlft
        xdef    sincntrgt
        xdef    sintstlft
        xdef    sintstrgt
        xdef    rngtbl
        xdef    xaxisincr
        xdef    thresh
        xdef    thresslb
        xdef    holdthresslb
        xdef    splitthresslb
        xdef    b_i
        xdef    fmap
        xdef    cb
        xdef    g_cb
        xdef    dbaddtbl
        xdef    curxlft
        xdef    curxrgt
        xdef    frmformat
```

```
        xdef    reedsolomon
        xdef    trailbits
        xdef    reedendpos
        xdef    psychaddr
        xdef    dbgaddr
        xdef    dbgflag

;tables of variables for sampling rate, framing bit rate and baud rate

        xdef    samplerates
        xdef    translaterates
        xdef    bitrates
        xdef    framevalues
        xdef    psychtable
        xdef    bauddata

        org     yhe:

stxcode_yhe

bitrate         ds      1       ;ISO frame header bit rate code as per frmrate
frmrate         ds      1       ;frame bit rate index for table manipulation
                                ; for either high or low sampling rate:
                                ;  code      high sampling   low sampling
                                ;    0           384             160
                                ;    1           256             144
                                ;    2           192             128
                                ;    3           128             112
                                ;    4           112              96
                                ;    5            96              80
                                ;    6            64              64
                                ;    7            56              56
                                ;    8           320              48
                                ;    9           224              40
                                ;   10           160              32
                                ;   11            80              24
                                ;   12            48              16
                                ;   13            32               8
                                ;   14(free)     399             399
rawrate         ds      1       ;raw input frame bit rate to be translated
                                ; switches (5 bits) indicate
                                ;         00000 = 384 Kbits
                                ;         00001 = 256 Kbits
                                ;         00010 = 192 Kbits
                                ;         00011 = 128 Kbits
                                ;         00100 = 112 Kbits
                                ;         00101 =  96 Kbits
                                ;         00110 =  64 Kbits
                                ;         00111 =  56 Kbits
                                ;         01000 = 320 Kbits
                                ;         01001 = 224 Kbits
                                ;         01010 = 160 Kbits
                                ;         01011 =  80 Kbits
                                ;         01100 =  48 Kbits
                                ;         01101 =  32 Kbits
                                ;         01110 = 144 Kbits
                                ;         01111 =  40 Kbits
                                ;         10000 =  24 Kbits
                                ;         10001 =  16 Kbits
                                ;         10010 =   8 Kbits
```

```
                                         ;       10011 = 399 Kbits (free bit rate)
smplrte         ds      1                ;audio sampling bit rate as to hardware
smplcde         ds      1                ;ISO frame hdr sample rate code as per smplrte
                                         ; switches (2 bits) indicate
                                         ;       00 = 44.1 K or 22.05 K
                                         ;       01 = 48 K or 24 K
                                         ;       10 = 32 K or 16 K
                                         ;       11 = CDQ1000 mono at 24 K sampling
smplidbit       ds      1                ;hdr id bit:
                                         ;       1 for 44.1, 48 and 32 K sample rates
                                         ;       0 for 22.05, 24 and 16 K sample rates
padrate         ds      1                ;frame padding calculation: sample rate
paddiff         ds      1                ;frame padding calc: DIFF @ sample/bit rates
padrest         ds      1                ;frame padding calculation: REST
usediff         ds      1                ;working diff for pad calculation
bndwdth         ds      1                ;code for setting sub-band limits
frmtype         ds      1                ;switches (2 bits) are set to:
                                         ;       00 = (0) full stereo
                                         ;       01 = (1) joint stereo
                                         ;       10 = (2) dual channel
                                         ;       11 = (3) mono (1 channel)
opfrtyp         ds      1                ;current frame type after bit allocation
                                         ; if unit coding joint stereo, the
                                         ;   frame could be full stereo as well
                                         ;   as joint stereo
maxsubbands     ds      1                ;MAXSUBBANDS for sample/bit rates
stintns         ds      1                ;intensity subband boundary code
oldccs          ds      1                ;encode MPEG-ISO vs old CCS CDQ2000's
                                         ;       0 = MPEG-ISO
                                         ;       1 = old CCS CDQ2000's
strtsinlft      ds      1                ;left channel -1 NOT sine, else 1st sub-band
endsinlft       ds      1                ;left channel -1 NOT sine, else 2nd sub-band
strtsinrgt      ds      1                ;right channel -1 NOT sine, else 1st sub-band
endsinrgt       ds      1                ;right channel -1 NOT sine, else 2nd sub-band
sincntlft       ds      1                ;sine test frame counter left channel
sincntrgt       ds      1                ;sine test frame counter right channel
sintstlft       ds      1                ;sine test flag left channel
sintstrgt       ds      1                ;sine test flag right channel
rngtbl                                   ;table for searching for tonals.
        dc      2,3,6,6,12,12,12,12
xaxisincr       ds      1                ;x axis increment for b_ii & ThresSLB tables
thresh          ds      1                ;threshold of hearing table choice for XPSYCHO
thresslb        ds      1                ;table address for current frame
holdthresslb    ds      1                ;normal frames table addr for current frame
splitthresslb   ds      1                ;mono split frames table addr for current frame
b_i             ds      1                ;table address for current frame
fmap            ds      1                ;table address for current frame
cb              ds      1                ;table address for current frame
g_cb            ds      1                ;table address for current frame
dbaddtbl        ds      1                ;table address for current frame
curxlft         ds      1                ;left channel-current location in x vector
curxrgt         ds      1                ;right channel-current location in x vector
frmformat       ds      1                ;communications frame formatting code

;Reed/Solomon frames controls:

reedsolomon     ds      1                ;Reed/Solomon switch (bit 0) 0=no y=yes
trailbits       ds      1                ;Reed/Solomon bits to take from end of frame
reedendpos      ds      1                ;Reed/Solomon bit count - frame flush zero bits
```

```
psychaddr        ds      1       ;addr psychtable as per current sampling rate
dbgaddr          ds      1      ;!!! debug save address
dbgflag          dc      0      ;!!! debug control flag
stsmplrts_yhe

        SAMPLERATES

endsmplrts_yhe
sttransl_yhe

        TRANSLATERATES

endtransl_yhe
stbitrts_yhe

        BITRATES

endbitrts_yhe
stfrmvals_yhe

        FRAMEVALUES

endfrmvals_yhe
stpsychtbl_yhe

        PSYCHTABLE

endpsychtbl_yhe
stbauddata_yhe

        BAUDDATA

endbauddata_yhe

endxcode_yhe

        endsec

        section ptable

        xdef    ptable
        xdef    a_psych,b_psych
        xdef    c_psych,d_psych
        xdef    e_psych,f_psych,g_psych
        xdef    h_psych,i_psych,j_psych
        xdef    k_psych,l_psych,m_psych,n_psych,o_psych,p_psych
        xdef    q_psych,r_psych,s_psych,t_psych,u_psych,v_psych,w_psych,x_psych
        xdef    y_psych,z_psych
        xdef    z1_psych,z2_psych,z3_psych,z4_psych,z5_psych,z6_psych

        org     yhe:
stptable_yhe

ptable

  ;his table is loaded with IRT factors from 12/92

a_psych          dc   0.0467146          ;A curval
b_psych          dc   0.0498289          ;B curval
c_psych          dc   0.0259526          ;C curval
```


BAD ORIGINAL

```
d_psych         dc   0.0498289          ;D curval
e_psych         dc   0.0882387          ;E curval
f_psych         dc   0.4000000          ;F curval
g_psych         dc   0.0311431          ;G curval
h_psych         dc   0.0882387          ;H curval
i_psych         dc   0.0882387          ;I curval
j_psych         dc   0.1000000          ;J curval
k_psych         dc   0.0000000          ;K curval
l_psych         dc   0.0000000          ;L curval
m_psych         dc   0.0000000          ;M curval
n_psych         dc   0.0000000          ;N curval
o_psych         dc   0.0000000          ;O curval
p_psych         dc   0.0000000          ;P curval
q_psych         dc   0.0000000          ;Q curval
r_psych         dc   0.0000000          ;R curval
s_psych         dc   0.0000000          ;S curval
t_psych         dc   0.0000000          ;T curval
u_psych         dc   0.0000000          ;U curval
v_psych         dc   0.0000000          ;V curval
w_psych         dc   0.0000000          ;W curval
x_psych         dc   0.0000000          ;X curval
y_psych         dc   0.0000000          ;Y curval
z_psych         dc   0.0000000          ;Z curval
z1_psych        dc   0.0000000          ;Z1 curval
z2_psych        dc   0.0000000          ;Z2 curval
z3_psych        dc   0.0000000          ;Z3 curval
z4_psych        dc   0.0000000          ;Z4 curval
z5_psych        dc   0.0000000          ;Z5 curval
z6_psych        dc   0.0000000          ;Z6 curval

e_adptable_yhe
        endsec

        org     phe:
start

; The external wait state is set to 1.  This allows the HCT541's to
; put their data on the bus in plenty of time.

        XCODE_M_BCR                     ;set all external io wait states

;set new crystal clock to 64 MHz

        XCODE_M_PCTL

;load X and Y memory

        jsr     boot_xy

;!!!dbg jsr     initdeb ;!!!debug init the debug port
;!!!dbg move    #>$720906,a     ;!!!debug
;!!!dbg jsr     outhex          ;!!!debug
;!!!dbg jsr     cr              ;!!!debug

; Clear all of the x memory

        clr     a               ;value to set x memory to
        move    #$ffff,m0       ;just in case, set to linear buffer
        move    #starty,r0      ;set starting address
        move    #(endy-starty),r1       ;set loop count
```


BAD ORIGINAL

```
        rep       r1                              ;clear r1
        move      a,y:(r0)+

; set the start of the block sequencing table for the 1st frame

        move      #seqnums,r1
        move      r1,y:nxtseq

;  PORT C Assignments
;
;  s = ssi port
;  i = input port
;  o = output port
;

        XCODE_PORT_C_M_PCC                ;set port C control register
        XCODE_PORT_C_M_PCD                ;set output data to port C
        XCODE_PORT_C_M_PCDDR              ;set port C data direction reg

; initialize the ssi port for the ad converter

        XCODE_SSI_M_CRA                   ;set ssi cra register
        XCODE_SSI_M_CRB                   ;set ssi crb register

; initialize the sci port for tty

        XCODE_SCI_M_SCR                   ;set sci status control register

;  PORT B Assignments

; 14 13 12 - 11 10 9 8 - 7 6 5 4 - 3 2 1 0
;  o  o  i     o  i i o   o i i i   i i i i
;

        XCODE_PORT_B_M_PBC        ;set B control register for general IO
        XCODE_PORT_B_M_PBD        ;set the default outputs
        XCODE_PORT_B_M_PBDDR      ;set B register direction

;initialize the AES-EBU chip

        XPSYCHO_AES_EBU_INIT

;initialize the host vactors

        INIT_HOST_VECTORS_CD

restart

; set the interrupt for host interrupts
; HOST set to IPL 2

        movep     #>$0800,x:<<M_IPR        ;set int priorities and edges
        andi      #$fc,mr                  ;turn on the interrupt system

;       ori       #$03,mr
        nop
        nop
        nop

;clear the analog to digital converter to restart calibration
```

```
        CLR_ADC_RESET

;disable the ancillary data received interrupt

        bclr    #M_RIE,x:<<M_SCR

;initialize the led applies word and light initial leds

        move    #>OFF_LEDS_CD,b         ;initialize leds as off
        move    b,y:<word_out
        ON_ALARM_LED_CD                 ;light alarm led indicator
        TST_SET_ALARM_RELAY_CD,_set_led_0       ;unless already set,
        SET_ALARM_RELAY_CD              ;set the alarm relay line on

_set_led_0

;see if an invalid bit rate was selected for the sampling rate

        OFF_INVALID_BIT_RATE_LED_CD     ;signal ok
        move    #smplidbit,r0           ;to test for bit rate error flag
        nop
        jclr    #4,y:(r0),_lite_leds    ;if no rate error, light the leds

;an invalid bit rate was selected for sampling rate

        ON_INVALID_BIT_RATE_LED_CD      ;signal the error

_lite_leds
        SET_LEDS_CD

;initialize the encoder control word: y:<stereo

        clr     a
        move    a,y:<stereo

;!!!dbg: initialize the debug counter

        move    a,y:dbgcnt

;!!!dbg

; init left and right channel start addresses in working x vector buffer
; and start initializing various registers for both channels

        move    #xbuflft,r0             ;left channel set start pos in x buffer
        move    r0,y:curxlft            ;save left channel start pos x buffer
        jsr     polyaini                ;left channel init poly analysis filter

        move    #xbufrgt,r0             ;right channel set start pos in x buffer
        move    r0,y:curxrgt            ;save right channel start pos x buffer
        jsr     polyaini                ;right channel init poly analysis filter

;initialize for joint framing intensity boundary control

        move    a,y:boundlst            ;zero last established boundary
        move    a,y:jfrmcnt             ;zero the frame count down ctl

;set up for receiving left and right channel PCM samples:
;  left channel values are stored on even addresses in the buffer with
```

```
;  right channel values stored in the adjacent odd buffer addresses

        move    #inpcm,r7                   ;get the input pcm data buffer
        move    r7,y:<ipwptr                ;set start left channel buffer address

;initialize for finding a sine wave as input

        move    a,x:<sincnt                 ;zero the sine frame counter
        move    a,y:sincntlft               ;zero left channel sine frame counter
        move    a,y:sincntrgt               ;zero right channel sine frame counter
        move    a,x:<sintest                ;clear the sine indicator
        move    a,y:sintstlft               ;clear the sine indicator left channel
        move    a,y:sintstrgt               ;clear the sine indicator right channel

;initialize the array of sub-band usage

        move    #UsedSBs,r0                 ;addr of used sub-band counters
        rep     #NUMSUBBANDS*NUMCHANNELS
        move    a,x:(r0)+

;indicate this is the 1st frame after a restart for scale factor checksum

        bclr    #2,x:private

;check the switches to determine bit rate and framing type
;  and ancillary data application and data baud rate

        GET_SWITCHES_CD gsws_00
        jsr     getsws

        move    x:tstrate,y1
        move    y1,y:rawrate                ;set the frame rate i/p code
        move    x:tstsmpl,y1
        move    y1,y:smplrte                ;set the sampling rate i/p code
        move    x:tstfrme,y1
        move    y1,y:frmtype                ;stereo, mono or joint frames
        move    x:tstband,y1
        move    y1,y:bndwdth                ;set allocation band width code
        move    x:tstbaud,y1
        move    y1,y:baudrte                ;set ancillary data baud rate code
        move    x:tstsel1,y1
        move    y1,x:select1                ;set whether or not line 1 selected
        move    x:tstsel2,y1
        move    y1,x:select2                ;set whether or not line 2 selected
        move    x:tstoccs,y1
        move    y1,y:oldccs                 ;set MPEG-ISO (0) or old CCS (1)
        move    x:tstfrmt,y1
        move    y1,y:frmformat              ;set the communication frame format
        move    x:tstreed,y1
        move    y1,y:reedsolomon            ;set Reed/Solomon frame format switch
;!!!    move    x:tstbits,y1
;!!!    move    y1,y:trailbits              ;bits taken from frame for Reed/Solomon

;set framing mode led

        move    y:frmtype,a                 ;get specified framing via switches
        move    a,y:opfrtyp                 ;set current frame type for output to
                                            ; the coded frame (this can change
                                            ; from frame to frame from JOINT_STEREO
                                            ; to FULL_STEREO if the JOINT_STEREO bit
```

```
                                        ; allocation applies and can handle the
                                        ; curr frame data as true full stereo)

        SET_FRAME_TYPE_LED_CD           ;light the proper leds

        move    y:frmtype,a             ;get specified framing via switches
        move    #>MONO,x0               ;start with mono
        cmp     x0,a    #>JOINT_STEREO,x0      ;compare and set up for JOINT
        jne     <_xcod_05

        OFF_JOINT_LED_CD                ;clear the JOINT stereo led
        OFF_STEREO_LED_CD               ;clear the FULL stereo led
        ON_MONO_LED_CD                  ;light the MONO led

;indicate mono framing (default is stereo)

        bset    #STEREO_vs_MONO,y:<stereo

        jmp     <_xcod_07

_xcod_05
        cmp     x0,a                    ;if not JOINT, defaults to full stereo
        jne     <_xcod_06

        OFF_MONO_LED_CD                 ;clear the MONO led
        OFF_STEREO_LED_CD               ;clear the FULL stereo led
        ON_JOINT_LED_CD                 ;light the JOINT stereo led

;indicate joint stereo framing (default is not joint)

        bset    #JOINT_FRAMING,y:<stereo

        jmp     <_xcod_07

_xcod_06
        OFF_MONO_LED_CD                 ;clear the MONO led
        OFF_JOINT_LED_CD                ;clear the JOINT stereo led
        ON_STEREO_LED_CD                ;light the FULL stereo led

_xcod_07
        SET_LEDS_CD

;based on the sampling rate and bit rate selected:
;       set the frame header sampling rate code
;       set the frame header bit rate code
;       set the frame size in words and bits
;       set the MAXSUBBANDS (for BALs)
;       set the applicable bit allocation control parameters

        move    #samplerates,r0         ;addr of sample rate values
        move    #DATABYSAMPLERATE,n0    ;num parameters per sample rate
        move    y:smplrte,b             ;to see if need to adjust address
        tst     b                       ;if code 0, no need to shift address
        jeq     <_smplrte               ;if 0, get the 3 parameters

;just the table address to proper sampling rate parameters

        rep     b
        move    (r0)+n0
```

```
_smplrte
        move    y:(r0)+,x0              ;get the ISO frame header ID code
        move    x0,y:smplidbit          ;save the ISO frame header ID code
        move    y:(r0)+,x0              ;get the ISO frame header code
        move    x0,y:smplcde            ;save the ISO frame header code
        move    y:(r0)+,x0              ;get the MAXCRITBNDS for XPSYCHO
        move    x0,y:<maxcritbnds       ;save the MAXCRITBNDS for XPSYCHO
        move    y:(r0)+,x0              ;get the NMSKFREQS for XPSYCHO
        move    x0,y:<nmskfreqs         ;save the NMSKFREQS for XPSYCHO
        move    y:(r0)+,x0              ;get the sample rate value for pad calc
        move    x0,y:padrate            ;save sample rate value for pad calc
        move    y:(r0)+,x0              ;get value to determine b_ii & ThresSLB
        move    x0,y:xaxisincr          ;save value for b_ii & ThresSLB tbls

;translate the raw bit rate code to the internal index rate code
;  based on whether the sampling rate is high (y:smplidbit 1=high) or low (0)
;and validate that the rate is supported by the software and/or hardware

        move    #translaterates,r0      ;addr of the translation table
        move    y:rawrate,n0            ;to offset to translated index
        bclr    #4,y:smplidbit          ;clear bad bit rate for sampling rate
        move    (r0)+n0                 ;pos to bit rate translate 1st value
        move    (r0)+n0                 ;pos to bit rate translate 2nd value
        move    y:smplidbit,n0          ;low (0) or high (1) sample rate select
        move    #>-1,a                  ;to see if not supported
        move    y:(r0+n0),x0            ;get the translated rate index code
        cmp     x0,a                    ;see if not supported rate
        jne     <_set_frmrate           ;if supported rate, set y:frmrate

 mpling rate does not support the selected bit rate

        bset    #4,y:smplidbit
        jmp     restart

_set_frmrate

;set the framing bit rate table index code

        move    x0,y:frmrate

;position to the proper set of framing bit rate parameters
;  based on high or low sampling rate selected

        move    #smplidbit,r1           ;to test for high sampling rate
        move    #bitrates,r0            ;addr of framing bit rate parms
        move    #BITRATESLOWOFFSET,n0   ;in case of low sampling rate
        jset    #0,y:(r1),_get_ISO      ;if high rate, continue
        move    (r0)+n0                 ;position to low sampling values

_get_ISO

;get the framing bit rate ISO header code and possible split rate parameters

        move    #DATABYBITRATE,n0       ;num parameters per bit rate code
        move    y:frmrate,b             ;to see if need to adjust address
        tst     b                       ;if code 0, no need to shift address
        jeq     <_bitrate               ;if 0, get the 3 parameters

;adjust the table address to proper bit rate parameters
```

```
        rep     b
        move    (r0)+n0

_bitrate
        move    y:(r0)+,x0              ;get the ISO frame header code
        move    x0,y:bitrate            ;save the ISO frame header code
        move    y:(r0)+,x0              ;get hi or lo rate threshold tbl ident
        move    x0,y:thresh             ;store hi or lo rate threshold tbl ident
        move    #0,n2                   ;init as not split rate capable
        move    #0,r2                   ;init as not split rate capable
        move    y:(r0)+,a               ;get optional split frame bit rate
        tst     a                       ;see if split rate applies
        jeq     <_setsplit              ;if not, set as null split rate parms

;indicate split frame mode of transmission can be used for this bit rate

        bset    #SPLIT_APPLIES,y:<stereo
        move    a,n2                    ;save optional split frame rate
        move    y:(r0)+,r2              ;get rate code for split rate band width

_setsplit

;set split rate parameters

        move    n2,y:spltrte            ;split mono frame rate code - header
        move    r2,y:spltbnd            ;split mono frame rate code - bandwidth

;get the framing parameters based on sampling rate and framing bit rate

        move    #framevalues,r0         ;addr of sample rate values
        move    #FRAMEBYSAMPLE,n0       ;numb parameters per sample rate
        move    y:smplrte,b             ;to see if need to adjust address
        tst     b                       ;if code 0, no need to shift address
        jeq     <_frbitrte              ;if 0, get the 3 parameters

;adjust the table address to proper sampling rate parameters

        rep     b
        move    (r0)+n0

_frbitrte
        move    #FRAMEBYBITRATE,n0      ;numb parameters per framing bit rate
        move    y:frmrate,b             ;test bit rate to set audio data size
        tst     b                       ;if code 0, no need to shift address
        jeq     <_frmdata               ;if 0, get the parameters

;adjust the table address to proper framing bit rate parameters at sample rate

        rep     b
        move    (r0)+n0

_frmdata
        move    y:(r0)+,x0              ;get the words per frame at rate
        move    x0,y:<outmus            ;save the words per frame at rate
        move    y:(r0)+,x0              ;get the bit count per frame at rate
        move    x0,y:<frmbits           ;save the bit count per frame at rate
        move    y:(r0)+,x0              ;get pad calc diff value @ samp/bit rtes
        move    x0,y:paddiff            ;save pad calc diff value
        move    x0,y:usediff            ;init as pad calc diff value
        move    (r0)+                   ;step over bit offset unpadded
```

```
        move    y:(r0)+,x0               ;get whether CCS compression applies
        move    x0,y:<cmprsctl           ;set CCS compression (1) or not (0)
        move    y:(r0)+,x0               ;get optional MAXSUBBANDS for split rate
        move    x0,y:spltmaxsubs         ;split mono frame rate MAXSUBBANDS
        move    y:(r0)+,x0               ;get split frame pad calc diff value
        move    x0,y:spltpaddiff         ;save split frame pad calc diff value

;set MAXSUBBANDS based on one or two channels coded at this bit rate

        move    #oldccs,r1               ;to test MPEG-ISO vs old CCS
        move    #0,n0                    ;set for two-channel MAXSUBBANDS
        jset    #0,y:(r1),_old_ccs       ;if set, do CCS the old way (use MONO)
        jclr    #STEREO_vs_MONO,y:<stereo,_maxsubs ;if 2 channel, offset is set

_old_ccs
        move    #1,n0                    ;set for one-channel MAXSUBBANDS
        nop

_maxsubs
        move    y:(r0+n0),x0             ;MAXSUBBANDS at rate and num channels
        move    x0,y:maxsubbands         ;save the MAXSUBBANDS at this rate
        move    x0,y:<maxsubs            ;set the working MAXSUBBANDS

;if old CCS switch is set.
;  see if CDQ1000 old mono frames at 24 K sampling and bit rate of 56 or 64

        move    #oldccs,r0               ;to see if old CCS requested
        move    #>SAM24K,x0              ;to check on 24 K sampling
        jclr    #0,y:(r0),_aft_old_CCS   ;if not old CCS requested, continue
        jclr    #STEREO_vs_MONO,y:<stereo,_aft_old_CCS  ;if 2 channel, continue
        move    y:smplrte,a              ;to test for 24 K sampling
        cmp     x0,a    #>RATE56_LOW,x0  ;see if sampling at 24 K
        jne     <_aft_old_CCS            ;if not continue
        move    y:frmrate,a              ;to test the framing bit rate
        move    #>BITRATE_56,x1          ;set high sampling rate bit rate code
        cmp     x0,a    #>RATE64_LOW,x0  ;see if 56 K frame rate
        jeq     <_set_cdq1000            ;if so, go to set up as if CDQ1000
        cmp     x0,a    #>BITRATE_64,x1  ;see if 64 K frame rate
                                         ; & set high sampling rate bit rate code
        jne     <_aft_old_CCS            ;if not, continue

_set_cdq1000

;we are doing a CDQ1000 mono frame at 24 K sampling

        move    #>SAMPLE_ID_BIT_HIGH,x0  ;set the frame header sampling id to 1
        move    x0,y:smplidbit           ;to insert 1 in the frame header
        move    #>SAMPLINGRATE_24_CDQ1K,x0    ;CDQ1000 sample rate code is 3
        move    x0,y:smplcde             ;set code for setsyst rtn
        move    #>27,x0                  ;MAXSUBBANDS = 27
        move    x0,y:maxsubbands         ;save the MAXSUBBANDS at this rate
        move    x0,y:<maxsubs            ;set the working MAXSUBBANDS
        move    x1,y:bitrate             ;set frame header bit rate code
        bset    #0,y:<cmprsctl           ;do CCS compression

_aft_old_CCS

;now set the type of ancillary count control:
;       0 = 3-bit pad byte count:
;                 CDQ2000 @ 48 K sampling
```

```
;               CDQ2001 & CDQ2012 @ 48 and 32 K sampling
;               CDQ2012 & CDQ1000 @ 24 K sampling as per old CCS CDQ's
;       1 = 8-bit data byte count:
;               frames coded @ 44.1 sampling rate
                frames coded with the sampling rate Id bit equal to 0
;                       MPEG-ISO @ 24, 22.05 and 16 K sampling
;               reed solomon frames

        clr     a       #smplidbit,r0   ;to init type of ancillary data count
                                        ; & set addr of sampling rate id bit
        move    a,y:anctype             ;init ancillary count type as old CCS
        jset    #0,y:(r0),_try_441      ;if not low sample rate id, try 44.1

;set the flag to output ancillary data byte count vs pad byte count

_set_data_cnt
        bset    #0,y:anctype            ;set to do data byte count
        jmp     <_set_psych_parms       ;continue: psychoacoustic parameters

_try_441
        move    y:smplrte,x0            ;to test for 44.1 sampling rate
        move    #>SAM44K,a              ;set 44.1 code
        cmp     x0,a    #reedsolomon,r0 ;see if sample rate is 44.1
                                        ; & set up to try reed solomon
        jeq     <_set_data_cnt          ;if 44.1, set data byte count type
        jset    #0,y:(r0),_set_data_cnt ;if reed solomon frames, data byte cnt

_set_psych_parms

  ased on the sampling rate from XCODE:
,       set the pyscho acoustic table of parameters

        move    #psychtable,r0          ;addr of psycho acoustic parameters
        move    #PSYCHBYSAMPLE,n0       ;num parameters per sample rate
        move    y:smplrte,b             ;to see if need to adjust address
        tst     b       #ptable,r1      ;if code 0, no need to shift address
                                        ; & set address of operational table
        jeq     <_ptable_copy           ;if 0, get the table parameters

;adjust the table address to proper sampling rate psycho acoustic parameters

        rep     b
        move    (r0)+n0

_ptable_copy

;save address of current sample rate psychtable for host vector update

        move    r0,y:psychaddr

;for the number of parameters copy sampling rate values to working table

        do      #PSYCHBYSAMPLE,_ptable_full
        move    y:(r0)+,x0
        move    x0,y:(r1)+

_ptable_full

;calculate buffer length controls
; for reed solomon set up for a three frame buffer for scale factor srs-12's
```


BAD ORIGINAL

```
        move    #reedsolomon,r4        ;to see if reed solomon applies
        move    y:<outmus,y1           ;get the words per frame
        move    #>2,x1                 ;standard is a 2 frame output buffer
        jclr    #0,y:(r4),_not_reed_a  ;if not reed solomon, 2 frame buffer
        move    #>3,x1                 ;for reed solomon make a 3 frame buffer

_not_reed_a
        mpy     x1,y1,a #>1,x1         ;set the mod buffer for numb frames
        asr     a                      ;align integer result
        move    a0,a                   ;shift integer result
        sub     x1,a                   ;(frame numb words * numb) - 1

;now save the above buffer control values

        move    a1,y:<outsize          ;set circular buffer ctl for o/p buffer

;set the type of frame as determined by the switches above

        move    #>BOUND_4,x0           ;stereo intensity default to 4
        move    x0,y:<sibound          ;save for frame header info
        move    #>INTENSITY_4,x0       ;stereo intensity code for default of 4
        move    x0,y:stintns           ;save for frame header info

;;;determine the type of framing STEREO vs MONO
;;
;;      move    y:z3_psych,y1          ;init with MONO band-width
;;      jset    #STEREO_vs_MONO,y:<stereo,_star_10 ;if mono, continue
;;      move    y:s_psych,y1           ;else, get FULL stereo band-width
        jclr    #JOINT_FRAMING,y:<stereo,_star_10 ;if joint bit allocation,
..      move    y:z4_psych,y1          ;else, get JOINT stereo band-width
;;
;;_star_10
;;      move    y1,y:<usedsb           ;set used sub-band width

;calculate the b_i, ThresSLB & Thres10SLB tables for the selected sampling rate
; build b_ii table

        move    #0,x0                  ;b_i starting value
        move    y:xaxisincr,x1         ;sample rate X-axis increment value
        move    #BarkX,r0              ;address of array of b_i X values
        move    #BarkY,r1              ;address of array of b_i Y values
        move    #BarkS,r2              ;address of array of b_i YP values
        move    #BrSScl,r5             ;address of b_i YP scale factor
        move    y:NBark,r6             ;number of points in look up table
        move    #>512,a                ;number of values to develop
        move    #b_ii,r3               ;address of b_ii table to be built
        jsr     mkbark                 ;build the b_i table

; build ThresSLB table

        move    #0,x0                  ;ThresSLB starting value
        move    y:xaxisincr,x1         ;sample rate X-axis increment value
        move    #0.0/192.66,y0         ;threshold adjustment in slb
        move    #ThresX,r0             ;address of array of ThresSLB X values
        move    #ThresY,r1             ;address of array of ThresSLB Y values
        move    #ThresS,r2             ;address of array of ThresSLB YP values
        move    #ThSScl,r5             ;address of ThresSLB YP scale factor
        move    y:NThres,r6            ;number of points in look up table
        move    #>512,a                ;number of values to develop
```


BAD ORIGINAL

```
        move    #ThresSLB,r3                ;address of ThresSLB table to be built
        jsr     mkthslb                     ;build the ThresSLB table

; build Thres10SLB table (10 dB down table)

        move    #0,x0                       ;ThresSLB starting value
        move    y:xaxisincr,x1              ;sample rate X-axis increment value
        move    #-12.04/192.66,y0           ;threshold adjustment in slb 10 dB down
        move    #ThresX,r0                  ;address of array of ThresSLB X values
        move    #ThresY,r1                  ;address of array of ThresSLB Y values
        move    #ThresS,r2                  ;address of array of ThresSLB YP values
        move    #ThSScl,r5                  ;address of ThresSLB YP scale factor
        move    y:NThres,r6                 ;number of points in look up table
        move    #>512,a                     ;number of values to develop
        move    #Thres10SLB,r3              ;address of Thres10SLB table to be built
        jsr     mkthslb                     ;build the Thres10SLB table

; set the proper XPSYCHO table addresses based on sampling rate

        move    y:smplrte,a                 ;get the sampling rate code
        move    #>SAM48K,x0                 ;to test for 48 K sampling
        cmp     x0,a      #>SAM44K,x0       ;test for 48 K
                                            ; & set up to test for 44.1 K sampling
        jeq     <_samp_48                   ;set up for 48 K sampling
        cmp     x0,a      #>SAM32K,x0       ;test for 44.1 K
                                            ; & set up to test for 32 K sampling
        jeq     <_samp_44                   ;set up for 44.1 K sampling
        cmp     x0,a      #>SAM24K,x0       ;test for 32 K
                                            ; & set up to test for 24 K sampling
        jeq     <_samp_32                   ;set up for 32 K sampling
        cmp     x0,a      #>SAM22K,x0       ;test for 24 K
                                            ; & set up to test for 22.05 K sampling
        jeq     <_samp_24                   ;set up for 24 K sampling
        cmp     x0,a      #>SAM16K,x0       ;test for 22.05 K
                                            ; & set up to test for 16 K sampling
        jeq     <_samp_22                   ;set up for 22.05 K sampling

_samp_16

; set up for 16 K Sampling

        move    #cb_16k,r4                  ;address of cb table @ 16 K sampling
        move    #g_cb_16k,r5                ;address of g_cb table @ 16 K sampling
        jmp     <_cont_00

_samp_22

; set up for 22.05 K Sampling

        move    #cb_22k,r4                  ;address of cb table @ 22.05 K sampling
        move    #g_cb_22k,r5                ;addr of g_cb table @ 22.05 K sampling
        jmp     <_cont_00

_samp_24

; set up for 24 K Sampling

        move    #cb_24k,r4                  ;address of cb table @ 24 K sampling
        move    #g_cb_24k,r5                ;address of g_cb table @ 24 K sampling
        jmp     <_cont_00
```

```
_samp_32

; set up for 32 K Sampling

        move    #cb_32k,r4              ;address of cb table @ 32 K sampling
        move    #g_cb_32k,r5            ;address of g_cb table @ 32 K sampling
        jmp     <_cont_00

_samp_44

; set up for 44.1 K Sampling

        move    #cb_44k,r4              ;address of cb table @ 44.1 K sampling
        move    #g_cb_44k,r5            ;address of g_cb table @ 44.1 K sampling
        jmp     <_cont_00

_samp_48

; set up for 48 K Sampling

        move    #cb_48k,r4              ;address of cb table @ 48 K sampling
        move    #g_cb_48k,r5            ;address of g_cb table @ 48 K sampling

_cont_00

; set the standard threshold of hearing tables

        move    #ThresSLB,r0

, set threshold of hearing table address as per switches bitrate & frame type

        move    y:thresh,a              ;get flag based on switches
        tst     a                       ;test the flag
        jne     <_cont_10               ;if non-zero, go with standard tables

; use the threshold of hearing table that is 10 db down

        move    #Thres10SLB,r0

_cont_10

;set the sampling rate table addresses for this frame

        move    r0,y:thresslb           ;set active selected table address
        move    r0,y:holdthresslb       ;save the selected table address
        move    #ThresSLB,x0
        move    x0,y:splitthresslb      ;set split mono table address
        move    #b_ii,r2                ;address of b_i table
        move    r2,y:b_i
        move    #fmap_x,r3              ;address of fmap table
        move    r3,y:fmap
        move    r4,y:cb
        move    r5,y:g_cb

 let output write read pointer to something safe since interrupts will
; be on before it is set properly.

        move    #framebuf,r0            ;address of the output frame buffer
        move    r0,y:<oprptr            ;set the output read buffer
```

```
;set up for ancillary data to be decoded from a framed and transmit via rs232
;       a. zero the input data byte counter and bytes for current frame
;       b. set address of clock table, raudclk, based on baud rate (0 thru 7)
;       c. set table offset by baud rate;
;          these are standard CDQ2000 set by macro, BAUDCLK, in box_ctl.asm):
;               0 = 300 baud
;               1 = 1200 baud
;               2 = 2400 baud
;               3 = 3200 baud
;               4 = 4800 baud
;               5 = 38400 baud
;               6 = 9600 baud
;               7 = 19200 baud
;       d. set transmit enable (for xon/xoff)
;       e. get and set the clock for baud rate from the table
;       f. get and set the max bytes for baud rate from the table
;       g. set the data input and output pointers
;       h. set receive enable
;       i. set receive enable interrupt

        move    #0,x0                   ;zero the received data counter
        move    x0,y:bytecnt            ;zero the byte counter
        move    x0,y:bytesfrm           ;zero the current frame byte counter
        move    #bauddata,r0            ;get data baud rate table address
        move    #DATABYBAUDRATE,n0      ;number parameters per baud rate
        move    y:baudrte,b             ;to see if need to adjust address
        tst     b                       ;if code 0, no need to shift address
        jeq     <_baudrte               ;if 0, get the clock parameter

;adjust the table address to proper ancillary data baud rate parameters

        rep     b
        move    (r0)+n0

_baudrte
        move    y:(r0)+,r2              ;get clock value at baud rate
                                        ; & move to code 0 max bytes per frame
        move    y:smplrte,n0            ;sample rate is now offset to max bytes
        nop
        move    y:(r0+n0),x0            ;frame max bytes for check of bytecnt
        move    x0,y:maxbytes           ;store maxbytes for scixmt to check
        move    #databytes,x0           ;get addr of the data byte buffer
        move    x0,y:dataiptr           ;address for next byte received
        move    x0,y:dataoptr           ;addr for next byte to output to frame
        movep   r2,x:<<M_SCCR           ;set the clock for selected baud rate
        bset    #M_RE,x:<<M_SCR         ;set receive enable
        bset    #M_RIE,x:<<M_SCR        ;data expected set receive interrupt
        bset    #M_TE,x:<<M_SCR         ;set transmit enable

;enable the host command interrupt

        bset    #M_HCIE,x:<<M_HCR

; Set and clear a flag so we can set the scope trigger.

        ON_BITALLOC_LED_CD              ;set a different flag for debug
        OFF_BITALLOC_LED_CD

; Now form the two pointers to the output buffer.
```

```
;       y:<frmstrt - is the write pointer for the 1st frame to be coded.
;       y:<oprptr - is the read pointer to start with the 2nd frame.
; frmstrt is used to point to where the current buffer is for outputting
; data into. This data is a result of the current musicam coding.

        move    #framebuf,r0            ;address of the output frame buffer
        move    y:<outmus,n0            ;set the output read ptr
        move    y:<outsize,m0           ;set the output buffer circular ctl
        move    r0,y:<frmstrt           ;1st frame at start of buffer
        move    (r0)+n0                 ;advance to start of 2nd frame
        move    r0,y:<frmnext           ;set to align with timer (frame sync)
        move    r0,y:<oprptr            ;set the output read buffer
        move    #$ffff,m0               ;reset to linear buffer

;start the padded frame REST variable at zero

        clr     a
        move    a,y:padrest

;start the output framing for setvalue at beginning of the frame buffer

        jsr     bitsallo

;clear the frame sync received flag and zero the time out counter

        bclr    #0,y:<timer             ;clear frame sync flag
        clr     a
        move    a,y:<timeout            ;zero frame syr: timed failure counter

  IRQA set to IPL 3, negative edge (lowest priority)
, SSI  set to IPL 3
; IRQB set to IPL 3, negative edge (highest priority)
; SCI  set to IPL 3
; HOST set to IPL 2
;all same priority

;       movep   #>$f03f,x:<<M_IPR       ;set int priorities and edges
        movep   #>$f83f,x:<<M_IPR       ;set int priorities and edges

;wait for the dust to settle before pushing onward

        move    #>XCODE_STARTUP,a
        jsr     wait

        SET_ADC_RESET                   ;stop A to D calibration

        movep   x:<<M_SR,a              ;clear the exception
        movep   #0,x:<<M_TX             ;output the data

        andi    #$fc,mr                 ;turn on the interrupt system

        stitle  'Main Code'
        page

;main loop capturing frames of audio, performing the psychoacoutinc analysis
  :d encoding MUSICAM frames of audio data that are sent to a MUSICAM decoder

top

;!!!dbg bset    WATCH_DOG               ;tickle the dog
```

```
;!!!dbg bclr    WATCH_DOG                   ;tickle the dog
        bset    #1,x:<<$FFE5        ;!!!dbg: WATCH_DOG
;       bclr    #1,x:<<$FFE5        ;!!!dbg: WATCH_DOG

  ;eck for time cut since last frame sync pulse

        move    y:<timeout,a                ;get top loop counter
        move    #>TOP_TIMEOUT_CD,x0         ;get failure value
        cmp     x0,a    #>1,x0              ;test counter versus failure value
                                            ; & set up to increment counter
        jlt     <_top_1                     ;if not failed, continue
        bclr    #1,y:<timer                 ;clear the 1st frame skipped flag
        jmp     restart                     ;restart and skip the 1st frame

_top_1

;increment the failure counter

        add     x0,a                        ;increment counter
        move    a,y:<timeout                ;save counter so far this loop

;get the external switches to determine if any changes that signal a restart

        GET_SWITCHES_CD gsws_10
        jsr     getsws
        jclr    #4,y:<not_appl,_lets_go
;!!!dbg
;       jmp     restart     ;!!! debug
;!!!dbg

;.e have to restart with new framing criteria,
; protect the decoding of frames by clearing 2 successive frame

        move    y:<frmstrt,r6               ;set starting for output buffer
        move    y:<outsize,m6               ;set the output buffer circular ctl
        clr     a       #reedsolomon,r4     ;to zero the output frames buffers
                                            ; & to see if reed solomon applies
        do      y:<outmus,_clear_1          ;clear the 1st frame
        move    a,y:(r6)+
_clear_1
        jclr    #0,y:<timer,_clear_1        ;check for new frame
        bclr    #0,y:<timer
        do      y:<outmus,_clear_2          ;clear the 2nd frame
        move    a,y:(r6)+
_clear_2
        jclr    #0,y:<timer,_clear_2        ;check for new frame
        bclr    #0,y:<timer
        jclr    #0,y:(r4),_clear_done       ;if not reed solomon, 2 frame buffer
        do      y:<outmus,_clear_3          ;clear the 3rd frame
        move    a,y:(r6)+
_clear_3
        jclr    #0,y:<timer,_clear_3        ;check for new frame
        bclr    #0,y:<timer
_clear_done
        move    #-1,m6                      ;restore r6 to linear buffer control
        jmp     restart                     ;let's start anew

_lets_go

;test to light AES-EBU led
```

```
        XPSYCHO_AES_EBU_TEST

;initialize stereo control settings to reflect current transmission

        jsr     setctls

        jclr    #0,y:<timer,top          ;check for new frame
        bclr    #0,y:<timer
        bclr    #0,y:<qtalloc            ;clear frame msec timer bit allocation

        bclr    #1,x:<<$FFE5    ;!!!dbg: WATCH_DOG

;zero the time out counter
;  and see if this is the first frame since last restart,
;    and if so, set 1st frame flag and restart again

        clr     a
        move    a,y:<timeout             ;reset the time out counter
        jset    #1,y:<timer,_lets_go_2   ;if 1st frame bypassed, continue
        bset    #1,y:<timer              ;indicate skipped the 1st frame
        jmp     restart

_lets_go_2

;toggle the host watch dog flag

        TOGGLE_WATCH_DOG_CL

;!!dbg: debug the encoding of frame when a frame count limit is reached
;       move    y:dbgcnt,a        ;!!!dbg: get debug frame count
;       move    #>1,x0            ;!!!dbg: to increment debug frame count
;       add     x0,a    #>40,x0   ;!!!dbg: incrment debug frame counter
;                                 ;!!!dbg:  & get count limit to start debugging
;       cmp     x0,a              ;!!!dbg: see if frame count reached limit
;       jlt     <_dbg_cont_0      ;!!!dbg: if not at limit, save new count
;!!!dbg: debug limit reached: turn off interrupts and encode what we've got now
;       ori     #$03,mr
;       nop
;       nop
;       nop
;       clr     a                 ;!!!dbg: zero the frame counter
;_dbg_cont_0
;       move    a,y:dbgcnt        ;!!!dbg: save new debug frame count value
;!!!dbg

;set the working value for padding calculation

        move    y:paddiff,x0             ;get normal DIFF value for pad calc
        move    x0,y:usediff             ;init as pad calc diff value

;set the joint boundary determination controls:
;       minimum joint sub-band to set boundary
;       left and right channel anti correlation tolerance value
;       minimum sub-band requiring at least 1 index allocation

        move    y:maxsubbands,x0         ;get normal MAXSUBBANDS
        move    x0,y:<maxsubs            ;set the working MAXSUBBANDS
        move    y:z3_psych,a             ;init with MONO band-width
        jset    #STEREO_vs_MONO,y:<stereo,_xxxx_10 ;if mono, continue
```

```
        move    y:s_psych,a                 ;else, get FULL stereo band-width
        jclr    #JOINT_FRAMING,y:<stereo,_xxxx_10 ;if joint bit allocation,
        move    y:z4_psych,a                ;else, get JOINT stereo band-width

  _xxx_10
        tst     a                           ;see if used sub-bands would be zero
        jeq     <_use_maxsubs               ;if zero, reset to maxsubbands
        cmp     x0,a      a,y:<usedsb       ;see if table sub-bands ok vs maxsubs
                                            ; & in case, set usedsb to table value
        jle     <_aft_10                    ;table value ok, continue

_use_maxsubs

;default used sub-band width to maxsubs

        move    x0,y:<usedsb

_aft_10

;calculate the b_i, ThresSLB & Thres10SLB tables for the selected sampling rate

        move    y:k_psych,x0                ;get minimum joint sub-band
        move    x0,y:jntmin                 ;set minimum for next frame
        move    y:l_psych,x0                ;get 2 channels anti correlation value
        move    x0,y:jntanti                ;set anti correlation tolerance value
        move    y:m_psych,x0                ;get minimum sub-band req 1 allocation
        move    x0,y:<limitsb               ;set LIMITSUBBANDS for next frame
        move    y:t_psych,x0                ;get joint frame decrement count value
        move    x0,y:jntfrms                ;set joint frame decrement count value

;set the selected DbAddTbl (@ 3db or 6db)

        move    y:v_psych,a                 ;get the selection variable
        move    #.5,x0                      ;get the test value
        cmp     x0,a      #DbAddTbl_3db,x0          ;less than half is 3db table
                                            ; & in case, set DbAdd table @ 3db
        jlt     <_cont_11                   ;if less, set the working table address

;the DbAdd table @ 6 db was selected

        move    #DbAddTbl_6db,x0

_cont_11

;set working table address for DbAddTbl

        move    x0,y:dbaddtbl

;if doing a split mode of transmission:
;       set framing controls
;       set the frame header bit rate code
;       set the frame size in words and bits
;       set the applicable bit allocation control parameters

        jclr    #SPLIT_MODE,y:<stereo,_top_60

        move    y:frmtype,a                 ;get specified framing via switches
        move    y:maxsubbands,b             ;get normal MAXSUBBANDS
        move    y:holdthresslb,y0           ;selected ThresSLB table addr
;;      move    y:holdthreshld,y1           ;selected Threshld table addr
```

```
;if we are doing a split mono frame, set the output frame type to mono

        jclr    #SPLIT_MONO_FRAME,y:<stereo,_top_05 ;if not appl, continue
        move    #>MONO,a
        move    y:spltmaxsubs,b            ;get split rate MAXSUBBANDS
        move    y:splitthresslb,y0         ;split mono ThresSLB table addr
        move    y:splitthreshld,y1         ;split mono Threshld table addr
;;      move    y:spltpaddiff,x0           ;get split DIFF value for pad calc
;???    move    x0,y:usediff               ;set DIFF value for pad calc
;???

_top_05
        move    a,y:opfrtyp                ;set current frame type for output to
                                           ; the coded frame (this can change
                                           ; from frame to frame from JOINT_STEREO
                                           ; to FULL_STEREO if the JOINT_STEREO bit
                                           ; allocation applies and can handle the
                                           ; curr frame data as true full stereo)
        move    b,y:<maxsubs               ;set working MAXSUBBANDS
        move    y0,y:thresslb              ;set active ThresSLB table addr
;;      move    y1,y:threshld              ;set active Threshld table addr

;initialize proper control flags:

        bclr    #STEREO_vs_MONO,y:<stereo ;default to stereo
        bclr    #JOINT_FRAMING,y:<stereo  ;default to NOT joint stere

        move    #>MONO,x0                 ;start with mono
        cmp     x0,a    #>JOINT_STEREO,x0 ;compare and set up for JOINT
        jne     <_top_10

;indicate mono framing (default is stereo)

        bset    #STEREO_vs_MONO,y:<stereo

        jmp     <_top_20

_top_10
        cmp     x0,a                       ;if not JOINT, defaults to full stereo
        jne     <_top_20

;indicate joint stereo framing (default is not joint)

        bset    #JOINT_FRAMING,y:<stereo

_top_20

;determine the sub-band ranges
;  if applicable, setup the 128 or 112 Kbits split frame mono

        jset    #SPLIT_MONO_FRAME,y:<stereo,_top_30

;otherwise, normal 128 or 112 frame

        move    y:s_psych,n0               ;num sub-bands for FULL STEREO
        move    y:z4_psych,n1              ;num sub-bands for JOINT STEREO
        move    y:z3_psych,n2              ;num sub-bands for MONO
        jmp     <_top_40

_top_30
```

```
;set the used sub-bands based on the split mono bit rate

        move    #11,n2              ;!!!dbg: for now force 11 usedsubbands

_top_40

;determine the type of framing STEREO vs MONO

        move    n2,a                            ;init with MONO band-width
        jset    #STEREO_vs_MONO,y:<stereo,_top_50 ;if stereo bit allocation,
        move    n0,a                            ; get FULL stereo band-width
        jclr    #JOINT_FRAMING,y:<stereo,_top_50 ;if joint bit allocation,
        move    n1,a                            ; get JOINT stereo band-width

_top_50
        tst     a        y:<maxsubs,x0    ;see if used sub-bands would be zero
                                          ; & get maxsubbands to test
        jeq     <_use_maxsubs_a           ;if zero, reset to maxsubbands
        cmp     x0,a     a,y:<usedsb      ;see if table sub-bands ok vs maxsubs
                                          ; & in case, set usedsb to table value
        jle     <_aft_10_a                ;table value ok, continue

_use_maxsubs_a

;default used sub-band width to maxsubs

        move    x0,y:<usedsb

_aft_10_a

_top_60

;start of XPSYCHO processing:

; start with the left channel:

        bclr    #LEFT_vs_RIGHT,y:<stereo

_chan_2nd_

;come back here to analyze the 2nd channel

; Now get the position to read the fft data from
; This buffer is offset from the polyphase filter to account for the
; delay through the filter.

        move    #PCMSIZE*2-1,m0           ;set as a mod buffer for both channels
        move    x:<polyst,r0              ;get input pcm buffer address

;test for need to adjust for 2nd channel

        jclr    #LEFT_vs_RIGHT,y:<stereo,_hann_00
        move    (r0)+                     ;advance to 2nd channel

_hann_00
        move    #(256-64),n0              ;back up to position fft
        move    #hbuf,r1                  ;get hanning output buffer address
        move    (r0)-n0                   ;back-up one channel
        move    (r0)-n0                   ;back-up another channel
```


BAD ORIGINAL

```
        move    #2,n0                   ;set offset for two channels
        jsr     hanning                 ;apply a hanning window
        move    #-1,m0                  ;restore r0 to linear buffer
        jsr     fft                     ;fft the data
;       move    #fftbuf,r0              ;real part of fft
;       move    #fftbuf,r4              ;imaginary part of fft
;       move    #power,r1               ;power array
        move    #hbuf,r0
        jsr     logpow                  ;compute power of fft data
        move    #power,r0               ;power array
        move    #SBMaxDb,r1             ;maximum in each sub-band (slb)
        move    #NUMSUBBANDS,n1         ;in case it's the 2nd channel
;test for left or right channel currently to set address for sub-band max values
        jclr    #LEFT_vs_RIGHT,y:<stereo,_sbmax_00
;adjust address for 2nd channel
        move    (r1)+n1                 ;offset to 2nd channel part of array
_sbmax_00
        jsr     findmaxi                ;find max power in a sub-band
        move    #power,r1               ;power array
        move    #Tonals,r2              ;tonal array
        move    #rngtbl,r4              ;range table for tonal search
        jsr     findtona                ;find tonals
        move    r3,x:<ntonals           ;save number of tonals
        move    #power,r1               ;power array
        move    #Tonals,r2              ;tonal array
        move    #rngtbl,r4              ;range table for tonal search
        jsr     zeropowe                ;zero power around tonals
        move    #power,r1               ;power array
        move    #NoisePwr,r2            ;address of the noise array
        jsr     findnois                ;find the noise
        move    #Maskers,r3             ;address of the masker structure
        move    #NoisePwr,r2            ;address of the noise array
        move    #Tonals,r1              ;address of the Tonals structure
        move    x:<ntonals,x0           ;# of tonals in Tonals structure
        jsr     mergemas                ;merge the maskers
        move    b,x:<nmasker            ;save # of maskers
        move    #Maskers,r0             ;address of the masker structure
        move    x:<nmasker,b            ;number of maskers in masker structure
        jsr     finddbma                ;find the db value of maskers
        move    #Maskers,r0             ;address of the masker structure
;       jsr     pruneclo                ;prune close maskers
        move    #Maskers,r0             ;address of the masker structure
        move    x:<nmasker,b            ;number of maskers in masker structure
```

```
        jsr     prunequi                ;prune quiet maskers

        move    #Maskers,r0             ;address of the masker structure
        move    x:<nmasker,b            ;number of maskers in masker structure
        jsr     prunemas                ;prune masked maskers

        jsr     findalis                ;find alising components

;if doing left or right channel currently to save number aliasers accordingly

        jset    #LEFT_vs_RIGHT,y:<stereo,_alis_00

        move    b,x:nalislft            ;save left channel in local memory
        jmp     <_alis_10

_alis_00
        move    b,x:nalisrgt            ;save right channel in local memory

_alis_10
        move    #Maskers,r4             ;address of the masker structure
        move    #GlbMsk,r1              ;address of global masking threshold
        move    #MAXNMSKFREQS,n1        ;in case it's the 2nd channel

;if left or right channel currently to set address for sub-band global mask

        jclr    #LEFT_vs_RIGHT,y:<stereo,_gbmsk_00

;adjust address for 2nd channel

        move    (r1)+n1                 ;offset to 2nd channel part of array

_gbmsk_00
        jsr     QCalcGlo                ;calculate global masking threshold

; test Maskers array for a sine wave

        move    #Maskers,r0             ;address of the masker structure
        jsr     tstsine                 ;check maskers for a sine wave

        move    x:<sincnt,y0            ;get the updated sine frame count
        move    x:<sintest,y1           ;get the updated sine flag

;test if doing left or right channel currently to save result of sine wave test

        jset    #LEFT_vs_RIGHT,y:<stereo,_sine_00

        move    x0,y:strtsinlft         ;save left channel in local memory
        move    x1,y:endsinlft          ;save left channel in local memory
        move    y0,y:sincntlft          ;save left channel frame counter
        move    y1,y:sintstlft          ;save left channel sine flag
        jmp     <_sine_10

_sine_00
        move    x0,y:strtsinrgt         ;save right channel in local memory
        move    x1,y:endsinrgt          ;save right channel in local memory
        move    a,y:sincntrgt           ;save right channel frame counter
        move    y1,y:sintstrgt          ;save right channel sine flag

_sine_10
```

```
; polyphase filter the input data

        move    x:<polyst,r0            ;get polyana start address-left channel
        move    #2,n0                   ;set offset for two channels
        move    #PCMSIZE*2-1,m0         ;set as a mod buffer for both channels
        move    y:curxlft,r3            ;left channel current x vector location
        move    #polydta,r5             ;left chan output buffer poly analyzed
        move    #INPCM,n5               ;in case of right channel

;test for left or right channel currently to set address for poly analyzed data

        jclr    #LEFT_vs_RIGHT,y:<stereo,_panal_00

        move    (r0)+                   ;advance start inpcm data to right chan
        move    y:curxrgt,r3            ;right channel curr x vector location

;adjust address for right channel

        move    (r5)+n5                 ;offset right channel poly analyzed data

_panal_00
        jsr     polyanal                ;poly analyze the data
        move    #-1,m0                  ;restore to a linear buffer

;if left or right channel currently to save address in x vector accordingly

        jset    #LEFT_vs_RIGHT,y:<stereo,_panal_20

;save left channel current position in x vector buffer

        move    r3,y:curxlft            ;left channel current x vector location
        jmp     <_panal_30

_panal_20

;save right channel current position in x vector buffer

        move    r3,y:curxrgt            ;right channel current x vector location

_panal_30

; develop the scale factors
;  initialize the table of scale factors to minimum amplitude (63 ==> 0 ampl)

        move    #SBndSKF,r1             ;left chan addr of subband scale factors
        move    #0,n0                   ;start with left channel polydata
        move    #0,n1                   ;start with left channel scale factors

;test for left or right channel currently to set address for scale factors

        jclr    #LEFT_vs_RIGHT,y:<stereo,_scfct_00

;adjust address for right channel

        move    #NUMSUBBANDS*NPERGROUP,n1 ;right channel scale factors
        move    #INPCM,n0               ;right channel poly analyzed data
        move    (r1)+n1                 ;offset to right channel part of array

_scfct_00
```

```
;initialize the scale factor array with the terminal value of 63

        move    #63,n4
        do      #NUMSUBBANDS*NPERGROUP,_init_00
        move    n4,x:(r1)+                ;store 63 in scale factor array

_init_00
        move    #polydta,r0               ;addr of poly analyzed data
        move    #SBndSKF,r1               ;addr of sub-band scale factors
        move    (r0)+n0                   ;offset to proper channel poly data
        move    (r1)+n1                   ;offset to proper channel scale factors
        jsr     findskf                   ;find scale factors

; develop the SBits for scale factors

        move    #SBndSKF,r0               ;left chan addr sub-band scale factors
        move    #SBits,r1                 ;left channel addr of sub-band sbits

;test for left or right channel to set addresses for scale factors and sbits

        jclr    #LEFT_vs_RIGHT,y:<stereo,_sbits_00

;adjust address for right channel

        move    #NUMSUBBANDS*NPERGROUP,n0 ;right channel scale factors
        move    #NUMSUBBANDS,n1           ;right channel scale factor sbits
        move    (r0)+n0                   ;offset to right channel part of array
        move    (r1)+n1                   ;offset to right channel part of array

_sbits_00
        jsr     pickskf                   ;pick the best scale factors

; set correct maximum level for the channel

        move    #SBndSKF,r0               ;left chan addr sub-band scale factors
        move    #SBMaxDb,r1               ;left maximum in each sub-band (slb)
        move    #polydta,r2               ;left chan poly analyzed buffer

;test for left or right channel to set addresses for scale factors and SBMaxDb

        jclr    #LEFT_vs_RIGHT,y:<stereo,_cksub_00

;adjust address for right channel

        move    #NUMSUBBANDS*NPERGROUP,n0 ;right channel scale factors
        move    #NUMSUBBANDS,n1           ;right channel scale factor sbits
        move    #INPCM,n2                 ;right channel poly analyzed data
        move    (r0)+n0                   ;offset to right channel part of array
        move    (r1)+n1                   ;offset to right channel part of array
        move    (r2)+n2                   ;offset to right channel part of array

_cksub_00

;determine which method to use to determine the sub-band maximum values

        move    y:u_psych,a               ;get use findrms.asm rtn parameter
        move    #.5,x1                    ;if less than .5, use checksub.asm rtn
        cmp     x1,a                      ;see if parameter less than .5
        jlt     <_do_checksub             ;if less, use checksub.asm rtn
```

```
;use RMS for maximum level for the sub-band

        move    r2,r0                   ;addr of poly analyzed data
        jsr     findrms                 ;find max in a subband
        jmp     <_set_min_mask          ;go to set minimum masking level

_do_checksub

; set correct maximum level for the channel

        jsr     checksub                ;find max in a subband

_set_min_mask

; set minimum masking level in each sub-band

        move    #MinMskDb,r1            ;minimum masking per subband (slb)
        move    #NUMSUBBANDS,n1         ;right channel minimum masking per subs

;test for left or right channel to set address

        jclr    #LEFT_vs_RIGHT,y:<stereo,_ckmin_00

;adjust address for right channel

        move    (r1)+n1                 ;offset to right channel part of array

_ckmin_00

        move    #GlbMsk,r0              ;global masking threshold
        jsr     findminm                ;find min masking

; set minimum masking level in each sub-band

        move    x:nalislft,a            ;left channel number of aliasers
        move    #Alising,r0             ;left channel aliasing structure
        move    #SBMaxDb,r1             ;left channel max in each sub-band (slb)

;test for left or right channel to set addresses

        jclr    #LEFT_vs_RIGHT,y:<stereo,_ckmax_00

        move    x:nalisrgt,a            ;right channel number of aliasers

;adjust address for right channel

        move    #MAXTONALS*ALIASSIZE,n0 ;right channel alias structures
        move    #NUMSUBBANDS,n1         ;right channel sub-band max's
        move    (r0)+n0                 ;offset to right channel part of array
        move    (r1)+n1                 ;offset to right channel part of array

_ckmax_00

        jsr     findmaxs                ;find the maximum signal

; we're doing mono frames,
;  skip the right channel XPSYCHO processing

        jset    #STEREO_vs_MONO,y:<stereo,_xcode_00 ; if doing mono, skip right
```

```
;if we've done the left channel, initialize and go back for the right channel

        jset    #LEFT_vs_RIGHT,y:<stereo,_xcode_00 ; if did right, continue

  ;t flag for right channel

        bset    #LEFT_vs_RIGHT,y:<stereo    ;indicate right channel
        jmp     <_chan_2nd_                 ;go back & do same stuff for right chan

_xcode_00

; We have now finished all XPSYCHO the processing.
;set the working frame length in bits and handle any required padded frame
;determination

        jsr     setframelen

;set number of fixed bits required, and the number of available bits for audio
;just in case we are doing JOINT framing, set the flag to determine the
;  fixed and available bits for full stereo

        bset    #JOINT_at_FULL,y:<stereo    ;to develop FULL bits available
        jsr     bitpool
        bclr    #JOINT_at_FULL,y:<stereo    ;clear after setting FULL bits
        move    x0,y:<fixbits               ;save fixed bit count
        move    x1,y:<audbits               ;save bit count available for alloc

;!!!dbg
;       nop
        nop
,       nop
;       nop
;       nop
;       jmp     top             ;!!! debug if using stored frames buffer
;!!!dbg

;test if a sine wave in either or both channels according to XPSYCHO's
; initialize as NOT a sine wave in either channel

        bclr    #LEFT_SINE_WAVE,y:<stereo
        bclr    #RIGHT_SINE_WAVE,y:<stereo

; if the starting sub-band == -1, NOT a sine wave

        move    #>-1,x0                     ;indicator NOT a sign wave
        move    y:strtsinlft,a              ;left sine wave start subband
        cmp     x0,a    y:endsinlft,y0      ;to see if not a sine wave
                                            ; & save Left EndSin
        jeq     <_sin_R1                    ;if not a sine wave, try right channel

;set left channel sine wave flag and light the indicating LED

        bset    #LEFT_SINE_WAVE,y:<stereo

_sin_R1

        jset    #STEREO_vs_MONO,y:<stereo,_sin_R2 ;1 chan, skip right channel

; if the starting sub-band == -1, NOT a sine wave
```

```
        move    y:strtsinrgt,b              ;right sine wave start subband
        cmp     x0,b    y:endsinrgt,y1      ;to see if not a sine wave
                                            ; & save right EndSin
        jeq     <_sin_R2                    ;if not a sine wave, do bit allocation

;set right channel sine wave flag and light the indicating LED

        bset    #RIGHT_SINE_WAVE,y:<stereo

_sin_R2

;if both chanels have a sine wave detected,
;       the audio input cannot be a test sine wave and must
;               be reset as true audio

        jclr    #LEFT_SINE_WAVE,y:<stereo,_sin_OK       ;NOT a sine wave, OK
        jclr    #RIGHT_SINE_WAVE,y:<stereo,_sin_OK      ;NOT a sine wave, OK

_kill_sin

; since both channels have a sine wave,
;       re-initialize as NOT a sine wave in either channel

        bclr    #LEFT_SINE_WAVE,y:<stereo
        bclr    #RIGHT_SINE_WAVE,y:<stereo
        move    #>-1,x0                     ;indicator NOT a sign wave
        move    x0,y:strtsinlft             ;set as not a sine wave
        move    x0,y:endsinlft              ;set as not a sine wave
        move    x0,y:strtsinrgt             ;set as not a sine wave
        move    x0,y:endsinrgt              ;set as not a sine wave

_sin_OK

;now see if there is a sine wave in one of the channels and if so,
;       see if the other channel has audio and if so, cancel the
;       sine wave in the one channel. The input does not qualify as
;       a test tone.

        jclr    #LEFT_SINE_WAVE,y:<stereo,_sin_RT

;left channel has a sine wave, get set to check for audio in right channel

        move    #SBndSKF,r0                 ;to get scale factor min-left
        move    #SBndSKF,r1                 ;to look for audio in right
        move    #NUMSUBBANDS*NPERGROUP,n1           ;offset to right
        move    y:strtsinlft,a
        move    y:endsinlft,b
        move    (r1)+n1                     ;to chk right channel for audio
        jclr    #RIGHT_SINE_WAVE,y:<stereo,_sin_1chan
        jmp     <_sin_OK2

_sin_RT
        jclr    #RIGHT_SINE_WAVE,y:<stereo,_sin_OK2

;right channel has a sine wave, get set to check for audio in left channel

        move    #SBndSKF,r0                 ;set scale factor array
        move    #NUMSUBBANDS*NPERGROUP,n0           ;offset to right channel
        move    #SBndSKF,r1                 ;to chk left channel for audio
        move    y:strtsinrgt,a
```


BAD ORIGINAL

```
        move     y:endsinrgt,b
        move     (r0)+n0                         ;to get scale factor min-right

_sin_1chan

;one channel has a sine wave, see if there is audio in the other channel

        tst      a         #NPERGROUP,n0         ;check for sub-band 0
                                                 ; & by 3 scale factors per sb
        jeq      <_sin_min                       ;if zero, go to get min skf

;advance to start sub-band of sine wave

        do       a,_sin_min
        move     (r0)+n0

_sin_min

;determine the number of sine wave scale facotrs to check for lowest

        sub      a,b       #>NPERGROUP,x0        ;calc number of sub-bands in range
                                                 ; & to calculate entries per sub-band
        move     #>1,y0                          ;to account for subtracted sub_band
        add      y0,b                            ;incrment 1 sub-band
        move     b,y0                            ;shift end sub-band to multiply reg
        mpy      x0,y0,a  #NPERGROUP,n1          ;calculate number of scale factors
                                                 ; & to skip sub-band 0 in other channel
        asr      a                               ;align the integer result

 et the lowest scale factor over the sine wave range

        move     #>63,b                          ;start with largest skf

;search over scale factor span from above

        do       a0,_get_min
        move     x:(r0)+,x0                      ;get the scale factor
        cmp      x0,b                            ;see if scale factor less than last
        tgt      x0,b                            ;if so, set new lower scale factor

_get_min

;calculate the minimum scale factor value to look for in the other channel
;        to see if audio is present

        move     #>SINE_SKF_TEST,x0              ;set the scale factor adjust to min
        add      x0,b                            ;calc the minimum scale factor to test
        move     (r1)+n1

        do       #NUMSUBBANDS*NPERGROUP-3,_sin_OK2
        move     x:(r1)+,x0                      ;get scale factor
        cmp      x0,b                            ;test versus minimum
        jlt      <_cont_sin_tst                  ;if not below minimum, not audio yet

;we have audion in the other channel, clear the sine wave indication

        enddo                                    ;break the loop, it's audio
        jmp      <_kill_sin                      ;clear the sine wave ind in one channe

_cont_sin_tst
```

```
        nop

_sin_OK2

  ; we got here, we have a legitmate test tone (sine wave) in one channel
        and nothing in the other channel

;allocate the bits in the frame by subband

        move    #SBits,r0                 ;scale factors
        move    #MinMskDb,r1              ;minimum masking per sub-band (slb)
        move    #SBMaxDb,r2               ;maximum in each sub-band (slb)
        move    #SBPos,r4                 ;sub-band position
        move    #SBIndx,r5                ;sub-band indicies
        jsr     bitalloc                  ;allocate the bits

        clr     a
        move    a,y:<bitscnt              ;start the bit counter of framed bits
        move    y:<frmstrt,r6             ;set starting for output buffer
        move    y:<outsize,m6             ;set the output buffer circular ctl

        jsr     setsync                   ;set the sync bits

        jsr     setsyst                   ;set the system bits

;set framing mode led

        move    y:opfrtyp,a               ;get current frame's type via bitalloc

        SET_FRAME_TYPE_LED_CD             ;light the proper leds

        move    y:opfrtyp,a               ;get current frame's type via bitalloc
        move    #>MONO,x0                 ;start with mono
        cmp     x0,a    #>JOINT_STEREO,x0         ;compare and set up for JOINT
        jne     <_xcde_55

        OFF_JOINT_LED_CD                  ;clear the JOINT stereo led
        OFF_STEREO_LED_CD                 ;clear the FULL stereo led
        ON_MONO_LED_CD                    ;light the MONO led
        jmp     <_xcde_57

_xcde_55
        cmp     x0,a                      ;if not JOINT, defaults to full stereo
        jne     <_xcde_56

        OFF_MONO_LED_CD                   ;clear the MONO led
        OFF_STEREO_LED_CD                 ;clear the FULL stereo led
        ON_JOINT_LED_CD                   ;light the JOINT stereo led
        jmp     <_xcde_57

_xcde_56
        OFF_MONO_LED_CD                   ;clear the MONO led
        OFF_JOINT_LED_CD                  ;clear the JOINT stereo led
        ON_STEREO_LED_CD                  ;light the FULL stereo led

_xcde_57
        SET_LEDS_CD

;if there is CRC-16 protection on the frame:
; set the CRC-16 checksum bit count for the old ISO method:
```

```
;  a. header bits covered by any type of frame
;       plus bits for the left channel also apply to any type of frame
;  b. set bits for possible right channel based on frame type
;  c. if not MONO, add bits for right channel
   d. save old ISO bit count for this frame
;  e. clear the space in the frame to zeroes

        jclr    #PROTECT,y:<stereo,_xcde_60     ;if no checksum, set allocations

        move    #>CRC_BITS_A+CRC_BITS_B,a
        move    #>CRC_BITS_B,x0             ;bit count for right channels
        jset    #STEREO_vs_MONO,y:<stereo,_xcde_58
        add     x0,a                        ;since its stereo, add for right channel

_xcde_58
        move    a,x:crcold                  ;set the old ISO CRC-16 bit count
        jsr     clrcrc                      ;clear the crc bits

_xcde_60

        move    #SBIndx,r0                  ;sub-band indicies
        jsr     setbal                      ;set the bit allocations

; if doing joint stereo,
;       from the intensity sub-band boundary thru the last used
;       sub-band, move the joint SBits, SKFs and the averaged poly samples
;       for setsbits, setskf and setdata routines
;       the SBits and SKfs are left and right channels
;       the poly samples are stored in the regular left channel samples

        jclr    #JOINT_FRAMING,y:<stereo,_xcde_699      ;Not Joint Framing
        jset    #JOINT_at_FULL,y:<stereo,_xcde_699      ;Joint upgraded to FULL

        move    #SBits,r0                   ;SBits array
        move    #JntSBits,r1                ;Joint stereo SBits array
        move    #SBndSKF,r2                 ;SKFs array
        move    #JntSBSKF,r3                ;Joint stereo SKFs array
        move    #polydta,r4                 ;addr of left chan poly analyzed data
        move    #JntPlAnal,r5               ;addr of joint chan poly analyzed data

        move    y:<sibound,x0               ;intensity stereo sub-band boundary
        move    y:<usedsb,a                 ;count of subbands used
        sub     x0,a        x0,n0           ;number of sub-bands to shift
                                            ; and position into reg SBits array
        clr     b           x0,n1           ;clear b register for accum
                                            ; and position into joint SBits array
        add     x0,b        ;x0,n4          ;step over SKF subbands by 1 of 3 pos
                                            ; and pos into reg poly samples array
        add     x0,b        ;x0,n5          ;step over SKF subbands by 2 of 3 pos
                                            ; and pos into joint poly samples array
        add     x0,b                        ;step over SKF subbands by 3 of 3 pos
        move    b1,n2                       ;adjust SKFs array to intesnity sub
        move    b1,n3                       ;adj joint SKFs array to intesnity sub

        move    (r0)+n0                     ;update SBits addr to boundary sub-band
        move    (r1)+n1                     ;update Joint SBits addr to boundary sb
        move    (r2)+n2                     ;upd SBndSKF addr to boundary sub-band
        move    (r3)+n3                     ;upd Joint SBndSKF addr to boundary sb
        move    r4,y0                       ;save starting addr Left poly samples
        move    r5,y1                       ;save starting addr Joint poly samples
```


BAD ORIGINAL

```
        move    #NUMSUBBANDS,n0                 ;to shift the right channel SBits
        move    n0,n1                           ;to shift the right channel JntSbits
        move    #NUMSUBBANDS*NPERGROUP,n2       ;to shift the right channel SBndSKF
        move    n2,n3                           ;to shift right channel Joint SBndSKF
        move    n0,n4                           ;to shift to next Left poly sample
        move    n0,n5                           ;to shift to next Joint poly sample

;shift for the number of subbands requiring the shift
;       1. the SBits are changed to the joint SBits
;       2. the scale factors are changed to the joint scale factors
;       3. the left channel Poly Samples are replaced with the averaged
;               joint Poly Samples (left + right)/2

        do      a,_xcde_69

;overlay the left and right SBits code

        move    x:(r1+n1),x0                    ;get right channel joint SBits code
        move    x0,x:(r0+n0)                    ;replace the regular right SBits code
        move    x:(r1)+,x0                      ;get left channel joint SBits code
        move    x0,x:(r0)+                      ;replace the regular left SBits code

;overlay the group of 3 scale factors per sub-band for left and right channels

        do      #NPERGROUP,_xcde_62
        move    x:(r3+n3),x0                    ;get right channel group SKF joint
        move    x0,x:(r2+n2)                    ;repl right channel group SKF regular
        move    x:(r3)+,x0                      ;get left channel group SKF joint
        move    x0,x:(r2)+                      ;repl left channel group SKF regular

                                                ;end of SKF shift loop current sub-band
_xcde_62
        nop

_xcde_69

;move the full stereo left channel samples up to the intensity sub-band
; boundary into the joint averaged samples ((left + right) / 2) array

        do      y:<sibound,_xcde_699

        move    y0,r4                           ;set current sub-band sample 0 position
        move    y1,r5                           ;set current sub-band sample 0 position

;shift the 36 samples for this sub-band

        do      #NUMPERSUBBAND*NPERGROUP,_xcde_66
        move    x:(r4)+n4,x0                    ;get left channel sample
        move    x0,x:(r5)+n5                    ;insert left sample into joint sample

_xcde_66

;set-up the starting sample address for the next sub-band

        move    y0,r4                           ;get current sub-band sample 0 position
        move    y1,r5                           ;get current sub-band sample 0 position
        move    (r4)+                           ;incr to next sub-band sample 0
        move    (r5)+                           ;incr to next sub-band sample 0
        move    r4,y0                           ;save starting addr
        move    r5,y1                           ;save starting addr
```

```
;continue with coding of SBits, SKfs and sample data

_xcde_699

;before doing anything else further, do the scale factor checksums
;and store in the end of the previous frame

        jsr     setckskf

;now continue coding the current frame

        move    #SBits,r0               ;SBits array
        move    #SBIndx,r1              ;sub-band indicies
        jsr     setsbits                ;set the sbits

        move    #SBndSKF,r0             ;scale factors
        move    #SBits,r1               ;SBits array
        move    #SBIndx,r2              ;sub-band indices
        jsr     setskf                  ;set the scale factors

        move    #SBPos,r3               ;sub-band allocated positions
        move    #SBndSKF,r2             ;scale factors
        move    #polydta,r0             ;to set addr right chan poly anal data
        move    #INPCM,n0               ;offset to right chan poly analyzed data
        move    #polydta,r1             ;addr of left chan poly analyzed data
        move    (r0)+n0                 ;addr of right chan poly analyzed data

;if doing joint, substitute the joint sample array for the left channel

        jclr    #JOINT_FRAMING,y:<stereo,_xcde_169      ;Not Joint Framing
        jset    #JOINT_at_FULL,y:<stereo,_xcde_169      ;Joint upgraded to FULL
        move    #JntPlAnal,r1           ;addr of joint left chan poly anal data

_xcde_169
        jsr     setdata                 ;set the data

;if protection CRC checksum is included, do the checksum calculation
;  and insert it into the frame following the header info

        jclr    #PROTECT,y:<stereo,_xcde_70 ;if 0, protection not applicable

        jsr     setcrc                  ;set the checksum

_xcde_70

        jsr     setancdata              ;output ancillary data

        jsr     bitsfree                ;flush remainder of bits to buffer

        move    #-1,m6                  ;restore r6 to linear buffer control

;signal to host

        INTERRUPT_HOST_CD

        jmp     top                     ;wait for the next frame

        end     start
```

```
;
;  c 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;

;       if SAMTYPE==SAM16K

;       define  MAXCRITBNDS_16K '21'

; noise masker positions
;cb

cb_16k
        dc    6   ; index  0
        dc    7   ; index  1
        dc    7   ; index  2
        dc    7   ; index  3
        dc    7   ; index  4
        dc    8   ; index  5
        dc    8   ; index  6
        dc    9   ; index  7
        dc   10   ; index  8
        dc   11   ; index  9
        dc   13   ; index 10
        dc   15   ; index 11
        dc   18   ; index 12
        dc   21   ; index 13
        dc   26   ; index 14
        dc   33   ; index 15
        dc   39   ; index 16
        dc   46   ; index 17
        dc   55   ; index 18
        dc   64   ; index 19
        dc   78   ; index 20
endcb_16k

; noise masker geometric position
;g_cb

g_cb_16k
        dc    1   ; index=  0,  freq(Hz)=   15.6
        dc    7   ; index=  1,  freq(Hz)=  109.4
        dc   15   ; index=  2,  freq(Hz)=  234.4
        dc   22   ; index=  3,  freq(Hz)=  343.8
        dc   29   ; index=  4,  freq(Hz)=  453.1
        dc   36   ; index=  5,  freq(Hz)=  562.5
        dc   44   ; index=  6,  freq(Hz)=  687.5
        dc   53   ; index=  7,  freq(Hz)=  828.1
        dc   62   ; index=  8,  freq(Hz)=  968.8
        dc   73   ; index=  9,  freq(Hz)= 1140.6
        dc   85   ; index= 10,  freq(Hz)= 1328.1
        dc   99   ; index= 11,  freq(Hz)= 1546.9
        dc  115   ; index= 12,  freq(Hz)= 1796.9
        dc  135   ; index= 13,  freq(Hz)= 2109.4
        dc  158   ; index= 14,  freq(Hz)= 2468.8
        dc  187   ; index= 15,  freq(Hz)= 2921.9
        dc  223   ; index= 16,  freq(Hz)= 3484.4
        dc  266   ; index= 17,  freq(Hz)= 4156.3
        dc  316   ; index= 18,  freq(Hz)= 4937.5
        dc  375   ; index= 19,  freq(Hz)= 5859.4
        dc  446   ; index= 20,  freq(Hz)= 6968.8
```

```
        dc 512  ; end of list indicator
endg_cb_15k
;       endif

;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;

;       if SAMTYPE==SAM22K

;       define  MAXCRITBNDS_22K '22'

; noise masker positions
;cb

cb_22k
        dc   5  ; index  0
        dc   4  ; index  1
        dc   5  ; index  2
        dc   5  ; index  3
        dc   6  ; index  4
        dc   5  ; index  5
        dc   6  ; index  6
        dc   7  ; index  7
        dc   7  ; index  8
        dc   8  ; index  9
        dc  10  ; index 10
        dc  11  ; index 11
        dc  12  ; index 12
        dc  16  ; index 13
        dc  19  ; index 14
        dc  23  ; index 15
        dc  29  ; index 16
        dc  33  ; index 17
        dc  40  ; index 18
        dc  47  ; index 19
        dc  56  ; index 20
        dc  72  ; index 21
endcb_22k

; noise masker geometric position
;g_cb

g_cb_22k
        dc   1  ; index=  0, freq(Hz)=   21.5
        dc   5  ; index=  1, freq(Hz)=  107.7
        dc  10  ; index=  2, freq(Hz)=  215.3
        dc  15  ; index=  3, freq(Hz)=  323.0
        dc  20  ; index=  4, freq(Hz)=  430.7
        dc  26  ; index=  5, freq(Hz)=  559.9
        dc  31  ; index=  6, freq(Hz)=  667.5
        dc  38  ; index=  7, freq(Hz)=  818.3
        dc  45  ; index=  8, freq(Hz)=  969.0
        dc  52  ; index=  9, freq(Hz)= 1119.7
        dc  61  ; index= 10, freq(Hz)= 1313.5
        dc  72  ; index= 11, freq(Hz)= 1550.4
        dc  83  ; index= 12, freq(Hz)= 1787.3
        dc  97  ; index= 13, freq(Hz)= 2088.7
```

```
        dc 115  ; index= 14, freq(Hz)= 2476.3
        dc 136  ; index= 15, freq(Hz)= 2928.5
        dc 161  ; index= 16, freq(Hz)= 3466.8
        dc 192  ; index= 17, freq(Hz)= 4134.4
        dc 229  ; index= 18, freq(Hz)= 4931.1
        dc 272  ; index= 19, freq(Hz)= 5857.0
        dc 323  ; index= 20, freq(Hz)= 6955.2
        dc 387  ; index= 21, freq(Hz)= 8333.3
        dc 512  ; end of list indicator

endg_cb_22k

;       endif

;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;

;       if SAMTYPE==SAM24K

;       define  MAXCRITBNDS_24K '23'

; noise masker positions
;cb

cb_24k
        dc    4  ; index  0
        dc    5  ; index  1
        dc    4  ; index  2
        dc    5  ; index  3
        dc    5  ; index  4
        dc    5  ; index  5
        dc    5  ; index  6
        dc    6  ; index  7
        dc    7  ; index  8
        dc    8  ; index  9
        dc    8  ; index 10
        dc   10  ; index 11
        dc   12  ; index 12
        dc   14  ; index 13
        dc   18  ; index 14
        dc   21  ; index 15
        dc   26  ; index 16
        dc   31  ; index 17
        dc   37  ; index 18
        dc   43  ; index 19
        dc   51  ; index 20
        dc   66  ; index 21
        dc   96  ; index 22
endcb_24k

; noise masker geometric position
;g_cb

g_cb_24k
        dc    1  ; index=  0, freq(Hz)=   23.4
        dc    5  ; index=  1, freq(Hz)=  117.2
        dc    9  ; index=  2, freq(Hz)=  210.9
        dc   14  ; index=  3, freq(Hz)=  328.1
        dc   19  ; index=  4, freq(Hz)=  445.3
```

```
        dc  24  ; index=  5, freq(Hz)=   562.5
        dc  29  ; index=  6, freq(Hz)=   679.7
        dc  34  ; index=  7, freq(Hz)=   796.9
        dc  41  ; index=  8, freq(Hz)=   960.9
        dc  48  ; index=  9, freq(Hz)=  1125.0
        dc  56  ; index= 10, freq(Hz)=  1312.5
        dc  65  ; index= 11, freq(Hz)=  1523.4
        dc  76  ; index= 12, freq(Hz)=  1781.3
        dc  89  ; index= 13, freq(Hz)=  2085.9
        dc 105  ; index= 14, freq(Hz)=  2460.9
        dc 125  ; index= 15, freq(Hz)=  2929.7
        dc 148  ; index= 16, freq(Hz)=  3468.8
        dc 176  ; index= 17, freq(Hz)=  4125.0
        dc 210  ; index= 18, freq(Hz)=  4921.9
        dc 250  ; index= 19, freq(Hz)=  5859.4
        dc 297  ; index= 20, freq(Hz)=  6960.9
        dc 355  ; index= 21, freq(Hz)=  8320.3
        dc 435  ; index= 22, freq(Hz)=10195.3
        dc 512  ; end of list indicator

endg_cb_24k

;       endif

;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;

;       if SAMTYPE==SAM32K

;       define  MAXCRITBNDS_32K '24'

; noise masker positions
;cb

cb_32k
        dc   3  ; index  0
        dc   4  ; index  1
        dc   3  ; index  2
        dc   3  ; index  3
        dc   4  ; index  4
        dc   4  ; index  5
        dc   4  ; index  6
        dc   5  ; index  7
        dc   5  ; index  8
        dc   5  ; index  9
        dc   7  ; index 10
        dc   7  ; index 11
        dc   9  ; index 12
        dc  11  ; index 13
        dc  13  ; index 14
        dc  16  ; index 15
        dc  19  ; index 16
        dc  24  ; index 17
        dc  27  ; index 18
        dc  32  ; index 19
        dc  39  ; index 20
        dc  49  ; index 21
        dc  72  ; index 22
        dc 129  ; index 23
```

```
endcb_32k

; noise masker geometric position
;g_cb

g_cb_32k
        dc   1   ; index=  0, freq(Hz)=   31.3
        dc   3   ; index=  1, freq(Hz)=   93.8
        dc   7   ; index=  2, freq(Hz)=  218.8
        dc  10   ; index=  3, freq(Hz)=  312.5
        dc  13   ; index=  4, freq(Hz)=  406.3
        dc  17   ; index=  5, freq(Hz)=  531.3
        dc  21   ; index=  6, freq(Hz)=  656.3
        dc  26   ; index=  7, freq(Hz)=  812.5
        dc  31   ; index=  8, freq(Hz)=  968.8
        dc  36   ; index=  9, freq(Hz)= 1125.0
        dc  42   ; index= 10, freq(Hz)= 1312.5
        dc  49   ; index= 11, freq(Hz)= 1531.3
        dc  57   ; index= 12, freq(Hz)= 1781.3
        dc  67   ; index= 13, freq(Hz)= 2093.8
        dc  79   ; index= 14, freq(Hz)= 2468.8
        dc  93   ; index= 15, freq(Hz)= 2906.3
        dc 111   ; index= 16, freq(Hz)= 3468.8
        dc 132   ; index= 17, freq(Hz)= 4125.0
        dc 157   ; index= 18, freq(Hz)= 4906.3
        dc 187   ; index= 19, freq(Hz)= 5843.8
        dc 222   ; index= 20, freq(Hz)= 6937.5
        dc 266   ; index= 21, freq(Hz)= 8312.5
        dc 326   ; index= 22, freq(Hz)=10187.5
        dc 423   ; index= 23, freq(Hz)=13218.8
        dc 512   ; end of list indicator

endg_cb_32k

;       endif

;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;

;       if SAMTYPE==SAM44K

;       define  MAXCRITBNDS_44K '24'

; noise masker positions
;cb

cb_44k
        dc   2   ; index  0
        dc   3   ; index  1
        dc   2   ; index  2
        dc   3   ; index  3
        dc   2   ; index  4
        dc   3   ; index  5
        dc   3   ; index  6
        dc   3   ; index  7
        dc   4   ; index  8
        dc   4   ; index  9
        dc   5   ; index 10
        dc   5   ; index 11
```

```
        dc   7  ; index 12
        dc   7  ; index 13
        dc  10  ; index 14
        dc  12  ; index 15
        dc  14  ; index 16
        dc  17  ; index 17
        dc  20  ; index 18
        dc  23  ; index 19
        dc  28  ; index 20
        dc  36  ; index 21
        dc  52  ; index 22
        dc  93  ; index 23
endcb_44k

; noise masker geometric position
;g_cb

g_cb_44k
        dc   0  ; index=  0, freq(Hz)=     0.0
        dc   2  ; index=  1, freq(Hz)=    86.1
        dc   4  ; index=  2, freq(Hz)=   172.3
        dc   7  ; index=  3, freq(Hz)=   301.5
        dc   9  ; index=  4, freq(Hz)=   387.6
        dc  12  ; index=  5, freq(Hz)=   516.8
        dc  15  ; index=  6, freq(Hz)=   646.0
        dc  18  ; index=  7, freq(Hz)=   775.2
        dc  21  ; index=  8, freq(Hz)=   904.4
        dc  25  ; index=  9, freq(Hz)=  1076.7
        dc  30  ; index= 10, freq(Hz)=  1292.0
        dc  35  ; index= 11, freq(Hz)=  1507.3
        dc  41  ; index= 12, freq(Hz)=  1765.7
        dc  48  ; index= 13, freq(Hz)=  2067.2
        dc  56  ; index= 14, freq(Hz)=  2411.7
        dc  67  ; index= 15, freq(Hz)=  2885.4
        dc  80  ; index= 16, freq(Hz)=  3445.3
        dc  96  ; index= 17, freq(Hz)=  4134.4
        dc 114  ; index= 18, freq(Hz)=  4909.6
        dc 136  ; index= 19, freq(Hz)=  5857.0
        dc 161  ; index= 20, freq(Hz)=  6933.7
        dc 193  ; index= 21, freq(Hz)=  8311.8
        dc 236  ; index= 22, freq(Hz)=10163.7
        dc 307  ; index= 23, freq(Hz)=13221.4
        dc 512  ; end of list indicator

endg_cb_44k

;       endif

;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;

;       if SAMTYPE==SAM48K

;       define  MAXCRITBNDS_48K '24'

; noise masker positions
;cb

cb_48k
```

```
        dc    2   ; index  0
        dc    2   ; index  1
        dc    3   ; index  2
        dc    2   ; index  3
        dc    2   ; index  4
        dc    3   ; index  5
        dc    3   ; index  6
        dc    3   ; index  7
        dc    3   ; index  8
        dc    4   ; index  9
        dc    4   ; index 10
        dc    5   ; index 11
        dc    6   ; index 12
        dc    7   ; index 13
        dc    9   ; index 14
        dc   11   ; index 15
        dc   13   ; index 16
        dc   15   ; index 17
        dc   18   ; index 18
        dc   22   ; index 19
        dc   26   ; index 20
        dc   33   ; index 21
        dc   48   ; index 22
        dc   85   ; index 23
endcb_48k

; noise masker geometric position
;g_cb

g_cb_48k
        dc   0   ; index=  0, freq(Hz)=     0.0
        dc   1   ; index=  1, freq(Hz)=    46.9
        dc   4   ; index=  2, freq(Hz)=   187.5
        dc   6   ; index=  3, freq(Hz)=   281.3
        dc   8   ; index=  4, freq(Hz)=   375.0
        dc  11   ; index=  5, freq(Hz)=   515.6
        dc  14   ; index=  6, freq(Hz)=   656.3
        dc  17   ; index=  7, freq(Hz)=   796.9
        dc  20   ; index=  8, freq(Hz)=   937.5
        dc  23   ; index=  9, freq(Hz)=  1078.1
        dc  27   ; index= 10, freq(Hz)=  1265.6
        dc  32   ; index= 11, freq(Hz)=  1500.0
        dc  37   ; index= 12, freq(Hz)=  1734.4
        dc  44   ; index= 13, freq(Hz)=  2062.5
        dc  52   ; index= 14, freq(Hz)=  2437.5
        dc  62   ; index= 15, freq(Hz)=  2906.3
        dc  74   ; index= 16, freq(Hz)=  3468.8
        dc  88   ; index= 17, freq(Hz)=  4125.0
        dc 104   ; index= 18, freq(Hz)=  4875.0
        dc 124   ; index= 19, freq(Hz)=  5812.5
        dc 148   ; index= 20, freq(Hz)=  6937.5
        dc 177   ; index= 21, freq(Hz)=  8296.9
        dc 217   ; index= 22, freq(Hz)=10171.9
        dc 282   ; index= 23, freq(Hz)=13218.8
        dc 512   ; end of list indicator

endg_cb_48k

;       endif
```

```
        opt     fc
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\compval.asm

;This routine is used to compress a 3 sample triple.
;It examines the values of each of the 3 samples and finds an alternate
;sequence of values which approximates the origional 3 samples.  The alternate
;sequence changes only the least significant bit of one of the 3 samples.

;For example, if the samples are quantized to 3 steps and the values of
;the samples are 0, 0, and 0, the the resulting 5 bit number is
;       0*9 + 0*3 + 0 = 0

;The 0,0,0 sequence is mapped into the sequence 0?1 since it was determined
;that the sequence 0,0,0 has a very low probability of occurance.

;The following tables have been done

;       3, 3 step values packed into 5 bits -> 4 bits
;       3, 5 step values packed into 7 bits -> 6 bits
;       3, 7 step values stored in   9 bits -> 8 bits

;In all cases, a triplett of values is considered. An entire block could have
;the same reduction but the tables would be large.  The current approach
;requires very little change to the ISO MUSICAM routine.

;The following must be done to incorporate the changes into MUSICAM.

;       1.  Tell the bit allocator that the positions 1, 2 and 3 corresponding
;               to the above 3 tables now have 4, 6 and 8 bits respectively
;               for each triplett instead of 5, 7 and 9 bits.

;       2.  Add a table lookup to convert a 5, 7, 9 bit value into
;               a 4, 6, 8 bit value respectively.

;The similar thing must be done in the decoder.

; on entry:
;       n4  - set to the number of bits sent to setvalue for encoding
;       n0  - set with the normally coded triplet value sent to setvalue
;               (this is used as index into the proper table)
; on exit:
;       y0 - contains the compressed value to replace the normal one
;               sent to setvalue
;
; destroyed:
;       r0
;       a

;this save variable for exclusive use by compval only

        section highmisc
        xdef    compvalR0Save

        org     xhe:
stcompval_xhe

compvalR0Save   ds      1
```


BAD ORIGINAL

```
endcompval_xhe
        endsec

        section compress
        xdef    table1
        xdef    table2
        xdef    table3

        org     yhe:

stcompress_yhe

;table1 compresses values from indexed bit allocation position 1 values

table1
        dc      0                       ; 0
        dc      0                       ; 1
        dc      0                       ; 2
        dc      1                       ; 3
        dc      2                       ; 4
        dc      3                       ; 5
        dc      1                       ; 6
        dc      2                       ; 7
        dc      3                       ; 8
        dc      4                       ; 9
        dc      5                       ; 10
        dc      5                       ; 11
        dc      6                       ; 12
        dc      7                       ; 13
        dc      8                       ; 14
        dc      9                       ; 15
        dc      9                       ; 16
        dc      10                      ; 17
        dc      11                      ; 18
        dc      12                      ; 19
        dc      13                      ; 20
        dc      11                      ; 21
        dc      12                      ; 22
        dc      13                      ; 23
        dc      14                      ; 24
        dc      14                      ; 25
        dc      14                      ; 26

;table2 compresses values from indexed bit allocation position 2 values

table2
        dc      0                       ; 0
        dc      0                       ; 1
        dc      1                       ; 2
        dc      7                       ; 3
        dc      7                       ; 4
        dc      0                       ; 5
        dc      0                       ; 6
        dc      1                       ; 7
        dc      1                       ; 8
        dc      1                       ; 9
        dc      2                       ; 10
        dc      2                       ; 11
        dc      3                       ; 12
```

```
dc      4      ; 13
dc      4      ; 14
dc      5      ; 15
dc      5      ; 16
dc      5      ; 17
dc      5      ; 18
dc      5      ; 19
dc      5      ; 20
dc      5      ; 21
dc      5      ; 22
dc      5      ; 23
dc      5      ; 24
dc      6      ; 25
dc      6      ; 26
dc      7      ; 27
dc      8      ; 28
dc      8      ; 29
dc      9      ; 30
dc      9      ; 31
dc      10     ; 32
dc      11     ; 33
dc      11     ; 34
dc      12     ; 35
dc      13     ; 36
dc      14     ; 37
dc      15     ; 38
dc      16     ; 39
dc      17     ; 40
dc      17     ; 41
dc      18     ; 42
dc      19     ; 43
dc      19     ; 44
dc      20     ; 45
dc      20     ; 46
dc      20     ; 47
dc      20     ; 48
dc      20     ; 49
dc      21     ; 50
dc      21     ; 51
dc      22     ; 52
dc      23     ; 53
dc      23     ; 54
dc      24     ; 55
dc      25     ; 56
dc      26     ; 57
dc      27     ; 58
dc      28     ; 59
dc      29     ; 60
dc      30     ; 61
dc      31     ; 62
dc      32     ; 63
dc      33     ; 64
dc      34     ; 65
dc      35     ; 66
dc      36     ; 67
dc      37     ; 68
dc      38     ; 69
dc      39     ; 70
dc      39     ; 71
dc      40     ; 72
```

```
	dc	41		; 73
	dc	41		; 74
	dc	42		; 75
	dc	42		; 76
	dc	42		; 77
	dc	42		; 78
	dc	42		; 79
	dc	43		; 80
	dc	43		; 81
	dc	44		; 82
	dc	45		; 83
	dc	45		; 84
	dc	46		; 85
	dc	47		; 86
	dc	48		; 87
	dc	49		; 88
	dc	50		; 89
	dc	51		; 90
	dc	51		; 91
	dc	52		; 92
	dc	53		; 93
	dc	53		; 94
	dc	54		; 95
	dc	54		; 96
	dc	55		; 97
	dc	56		; 98
	dc	56		; 99
	dc	57		; 100
	dc	57		; 101
	dc	57		; 102
	dc	57		; 103
	dc	57		; 104
	dc	57		; 105
	dc	57		; 106
	dc	57		; 107
	dc	57		; 108
	dc	57		; 109
	dc	58		; 110
	dc	58		; 111
	dc	59		; 112
	dc	60		; 113
	dc	60		; 114
	dc	61		; 115
	dc	61		; 116
	dc	61		; 117
	dc	62		; 118
	dc	62		; 119
	dc	54		; 120
	dc	54		; 121
	dc	61		; 122
	dc	62		; 123
	dc	62		; 124

;table3 compresses values from indexed bit allocation position 3 values

table3
	dc	22		; 0
	dc	22		; 1
	dc	1		; 2
	dc	1		; 3
```

```
dc      2           ; 4
dc      2           ; 5
dc      7           ; 6
dc      0000        ; unused
dc      0           ; 8
dc      3           ; 9
dc      0           ; 10
dc      1           ; 11
dc      2           ; 12
dc      2           ; 13
dc      7           ; 14
dc      0000        ; unused
dc      8           ; 16
dc      3           ; 17
dc      3           ; 18
dc      4           ; 19
dc      5           ; 20
dc      6           ; 21
dc      7           ; 22
dc      0000        ; unused
dc      8           ; 24
dc      8           ; 25
dc      9           ; 26
dc      10          ; 27
dc      11          ; 28
dc      12          ; 29
dc      13          ; 30
dc      0000        ; unused
dc      14          ; 32
dc      14          ; 33
dc      15          ; 34
dc      16          ; 35
dc      17          ; 36
dc      18          ; 37
dc      18          ; 38
dc      0000        ; unused
dc      52          ; 40
dc      19          ; 41
dc      19          ; 42
dc      20          ; 43
dc      21          ; 44
dc      21          ; 45
dc      57          ; 46
dc      0000        ; unused
dc      58          ; 48
dc      58          ; 49
dc      19          ; 50
dc      20          ; 51
dc      21          ; 52
dc      21          ; 53
dc      57          ; 54
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
```

```
dc      22      ; 64
dc      22      ; 65
dc      23      ; 66
dc      24      ; 67
dc      25      ; 68
dc      30      ; 69
dc      30      ; 70
dc      0000    ; unused
dc      26      ; 72
dc      26      ; 73
dc      27      ; 74
dc      28      ; 75
dc      29      ; 76
dc      30      ; 77
dc      30      ; 78
dc      0000    ; unused
dc      31      ; 80
dc      32      ; 81
dc      33      ; 82
dc      34      ; 83
dc      35      ; 84
dc      36      ; 85
dc      37      ; 86
dc      0000    ; unused
dc      38      ; 88
dc      39      ; 89
dc      40      ; 90
dc      41      ; 91
dc      42      ; 92
dc      43      ; 93
dc      44      ; 94
dc      0000    ; unused
dc      45      ; 96
dc      46      ; 97
dc      47      ; 98
dc      48      ; 99
dc      49      ; 100
dc      50      ; 101
dc      51      ; 102
dc      0000    ; unused
dc      52      ; 104
dc      53      ; 105
dc      54      ; 106
dc      55      ; 107
dc      56      ; 108
dc      57      ; 109
dc      57      ; 110
dc      0000    ; unused
dc      58      ; 112
dc      58      ; 113
dc      59      ; 114
dc      60      ; 115
dc      61      ; 116
dc      61      ; 117
dc      57      ; 118
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
```

```
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      62          ; 128
dc      62          ; 129
dc      63          ; 130
dc      64          ; 131
dc      65          ; 132
dc      66          ; 133
dc      66          ; 134
dc      0000        ; unused
dc      67          ; 136
dc      68          ; 137
dc      69          ; 138
dc      70          ; 139
dc      71          ; 140
dc      72          ; 141
dc      73          ; 142
dc      0000        ; unused
dc      74          ; 144
dc      75          ; 145
dc      76          ; 146
dc      77          ; 147
dc      78          ; 148
dc      79          ; 149
dc      80          ; 150
dc      0000        ; unused
dc      81          ; 152
dc      82          ; 153
dc      83          ; 154
dc      84          ; 155
dc      85          ; 156
dc      86          ; 157
dc      87          ; 158
dc      0000        ; unused
dc      88          ; 160
dc      89          ; 161
dc      90          ; 162
dc      91          ; 163
dc      92          ; 164
dc      93          ; 165
dc      94          ; 166
dc      0000        ; unused
dc      95          ; 168
dc      96          ; 169
dc      97          ; 170
dc      98          ; 171
dc      99          ; 172
dc      100         ; 173
dc      101         ; 174
dc      0000        ; unused
dc      102         ; 176
dc      102         ; 177
dc      103         ; 178
dc      104         ; 179
dc      105         ; 180
dc      106         ; 181
dc      106         ; 182
dc      0000        ; unused
```

```
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      0000        ; unused
dc      107         ; 192
dc      107         ; 193
dc      108         ; 194
dc      109         ; 195
dc      110         ; 196
dc      111         ; 197
dc      118         ; 198
dc      0000        ; unused
dc      112         ; 200
dc      113         ; 201
dc      114         ; 202
dc      115         ; 203
dc      116         ; 204
dc      117         ; 205
dc      118         ; 206
dc      0000        ; unused
dc      119         ; 208
dc      120         ; 209
dc      121         ; 210
dc      122         ; 211
dc      123         ; 212
dc      124         ; 213
dc      125         ; 214
dc      0000        ; unused
dc      126         ; 216
dc      127         ; 217
dc      128         ; 218
dc      129         ; 219
dc      130         ; 220
dc      131         ; 221
dc      132         ; 222
dc      0000        ; unused
dc      133         ; 224
dc      134         ; 225
dc      135         ; 226
dc      136         ; 227
dc      137         ; 228
dc      138         ; 229
dc      139         ; 230
dc      0000        ; unused
dc      140         ; 232
dc      141         ; 233
dc      142         ; 234
dc      143         ; 235
dc      144         ; 236
dc      145         ; 237
dc      146         ; 238
dc      0000        ; unused
dc      140         ; 240
dc      147         ; 241
dc      147         ; 242
dc      148         ; 243
```

```
dc      149     ; 244
dc      150     ; 245
dc      150     ; 246
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      151     ; 256
dc      151     ; 257
dc      152     ; 258
dc      153     ; 259
dc      154     ; 260
dc      155     ; 261
dc      155     ; 262
dc      0000    ; unused
dc      156     ; 264
dc      157     ; 265
dc      158     ; 266
dc      159     ; 267
dc      160     ; 268
dc      161     ; 269
dc      162     ; 270
dc      0000    ; unused
dc      163     ; 272
dc      164     ; 273
dc      165     ; 274
dc      166     ; 275
dc      167     ; 276
dc      168     ; 277
dc      169     ; 278
dc      0000    ; unused
dc      170     ; 280
dc      171     ; 281
dc      172     ; 282
dc      173     ; 283
dc      174     ; 284
dc      175     ; 285
dc      176     ; 286
dc      0000    ; unused
dc      177     ; 288
dc      178     ; 289
dc      179     ; 290
dc      180     ; 291
dc      181     ; 292
dc      182     ; 293
dc      183     ; 294
dc      0000    ; unused
dc      184     ; 296
dc      185     ; 297
dc      186     ; 298
dc      187     ; 299
dc      188     ; 300
dc      189     ; 301
dc      190     ; 302
dc      0000    ; unused
```

```
dc        191        ; 304
dc        191        ; 305
dc        192        ; 306
dc        193        ; 307
dc        194        ; 308
dc        195        ; 309
dc        195        ; 310
dc        0000       ; unused
dc        0000       ; unused
dc        0000       ; unused
dc        0000       ; unused
dc        0000       ; unused
dc        0000       ; unused
dc        0000       ; unused
dc        0000       ; unused
dc        0000       ; unused
dc        200        ; 320
dc        196        ; 321
dc        196        ; 322
dc        197        ; 323
dc        198        ; 324
dc        199        ; 325
dc        199        ; 326
dc        0000       ; unused
dc        200        ; 328
dc        201        ; 329
dc        202        ; 330
dc        203        ; 331
dc        204        ; 332
dc        205        ; 333
dc        205        ; 334
dc        0000       ; unused
dc        206        ; 336
dc        207        ; 337
dc        208        ; 338
dc        209        ; 339
dc        210        ; 340
dc        211        ; 341
dc        211        ; 342
dc        0000       ; unused
dc        212        ; 344
dc        213        ; 345
dc        214        ; 346
dc        215        ; 347
dc        216        ; 348
dc        217        ; 349
dc        217        ; 350
dc        0000       ; unused
dc        218        ; 352
dc        219        ; 353
dc        220        ; 354
dc        221        ; 355
dc        222        ; 356
dc        223        ; 357
dc        223        ; 358
dc        0000       ; unused
dc        224        ; 360
dc        224        ; 361
dc        225        ; 362
dc        226        ; 363
```

```
dc      227     ; 364
dc      228     ; 365
dc      228     ; 366
dc      0000    ; unused
dc      224     ; 368
dc      224     ; 369
dc      229     ; 370
dc      229     ; 371
dc      230     ; 372
dc      231     ; 373
dc      231     ; 374
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      0000    ; unused
dc      200     ; 384
dc      233     ; 385
dc      233     ; 386
dc      234     ; 387
dc      232     ; 388
dc      199     ; 389
dc      199     ; 390
dc      0000    ; unused
dc      200     ; 392
dc      233     ; 393
dc      233     ; 394
dc      234     ; 395
dc      235     ; 396
dc      235     ; 397
dc      205     ; 398
dc      0000    ; unused
dc      236     ; 400
dc      236     ; 401
dc      237     ; 402
dc      238     ; 403
dc      239     ; 404
dc      239     ; 405
dc      239     ; 406
dc      0000    ; unused
dc      240     ; 408
dc      241     ; 409
dc      242     ; 410
dc      243     ; 411
dc      244     ; 412
dc      245     ; 413
dc      245     ; 414
dc      0000    ; unused
dc      246     ; 416
dc      246     ; 417
dc      247     ; 418
dc      248     ; 419
dc      249     ; 420
dc      250     ; 421
dc      254     ; 422
dc      0000    ; unused
```

```
        dc      246                 ; 424
        dc      251                 ; 425
        dc      251                 ; 426
        dc      252                 ; 427
        dc      253                 ; 428
        dc      254                 ; 429
        dc      254                 ; 430
        dc      0000                ; unused
        dc      246                 ; 432
        dc      251                 ; 433
        dc      251                 ; 434
        dc      252                 ; 435
        dc      253                 ; 436
        dc      231                 ; 437
        dc      231                 ; 438
        dc      0000                ; unused

endcompress_yhe

        endsec

        org     phe:

compval
        move    r0,x:compvalR0Save          ;save the register

;test the number of bits to choose the proper table:
; 4 bits - corresponds to table 1 with a 4 bit coded value
; 6 bits - corresponds to table 2 with a 6 bit coded value
; 8 bits - corresponds to table 3 with a 8 bit coded value

        move    n4,a
        move    #>4,y0                      ;test for table 1 first
        cmp     y0,a    #>6,y0              ;is table 1 chosen
                                            ; & set up for testing for table 2
        jeq     _cval_20                    ;if eq, go set proper table address

        cmp     y0,a                        ;is table 2 chosen
        jeq     _cval_10                    ;if eq, go set proper table address

        move    #table3,r0                  ;must be table 3, set its address
        jmp     _cval_30

_cval_10
        move    #table2,r0                  ;set address of table 2
        jmp     _cval_30

_cval_20
        move    #table1,r0                  ;set address of table 1

_cval_30

        nop
        move    y:(r0+n0),y0                ;return the compressed value
        move    x:compvalR0Save,r0          ;restore the register
        nop
        rts
```

```
        opt     fc
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\setsyst.asm

        title   'Set the system word'

; This routine outputs the 20-bit system header information in tothe frame.
; The frame header immediately follows the 12 bit sync word.

; on entry
;       r6 = current offset in output array
;       y:sc = shift count

; on exit
;       a = destroyed
;       b = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r4 = destroyed
;       n4 = destroyed
;

        include 'def.asm'
        include 'box_ctl.asm'

        org     phe:

setsyst

;bits 0 thru 3 of MUSICAM frame header:
;           0 = ID: high (1) or low (0) sampling rate
;         1-2 = '10' idetifies frame as MPEG-ISO Layer II
;           3 = CRC-16 protected: YES (0) or NO (1)

        move    #smplidbit,r0              ;get addr of high/low sample rate id
        jclr    #PROTECT,y:<stereo,_syst_00 ;protection does not apply if 0
        move    #>SYSTHDR_1_PROTECT,y0     ;bits 0-3 of frame header with CRC
        jset    #0,y:(r0),_syst_10         ;if high sample rate id, continue
        move    #>SYSTHDR_1_PROTECT_LOW,y0  ;replace with header id for low
        jmp     _syst_10

_syst_00
        move    #>SYSTHDR_1_NO_PROT,y0     ;bits 0-3 of frame header with CRC
        jset    #0,y:(r0),_syst_10         ;if high sample rate id, continue
        move    #>SYSTHDR_1_NO_PROT_LOW,y0  ;replace with header id for low

_syst_10

;output frame header bits 0 thru 3

        move    #NSYSTHDR_1,n4             ;number of bits
        jsr     <setvalue                  ;output the value

;bits 4 thru 7 of MUSICAM frame header: bit rate set as per dip switches

        move    y:bitrate,y0               ;bits 4-7 of frame header
        move    #NBITRATE,n4               ;number of bits
```

```
;if a CDQ2000 split rate mono frame, switch bit rate in frame header

        jclr    #SPLIT_MONO_FRAME,y:<stereo,_syst_20
        move    y:spltrte,y0                ;bits 4-7 of frame header

_syst_20

;output frame header bits 4 thru 7

        jsr     <setvalue                   ;output the value

;bits 8 and 9 of MUSICAM frame header: sampling rate

        move    y:smplcde,y0                ;bits 8-9 of frame header
        move    #NSAMPLERATE,n4             ;number of bits
        jsr     <setvalue                   ;output the value

;bits 10 and 11 of MUSICAM frame header:
;         10 = padding bit: 0-no padding bits 1-8 padding bits
;         11 = privacy bit: as set by user
; test if the frame is padded or not with 8 added bits

        clr     a                           ;to initialize bits 10 and 11
        move    a,x:tstfrme                 ;temp variable to set the bits
        move    y:usediff,a                 ;tst if padded frame code needed
        tst     a                           ;see if frame not padded (a = 0)
        jeq     _syst_30                    ;the padding bit is already set to 0

;frame is padded with 8 additional bits

        bset    #1,x:tstfrme                ;bit 10 set for padded frame

_syst_30

;set privacy bit as per user input

        TST_CLR_HEADER_BIT_11_CD,_syst_40       ;if not 0, continue
        bset    #0,x:tstfrme                ;set the privacy bit

_syst_40

;output frame header bits 10 and 11

        move    x:tstfrme,y0                ;formatted bits
        move    #NSYSTHDR_2,n4              ;number of bits
        jsr     <setvalue                   ;output the value

;bits 12 and 13 of MUSICAM frame header: mode
;               full stereo, joint stereo, dual channel or mono

        move    y:opfrtyp,y0                ;bits 12-13 of frame header
        move    #NFRAMETYPE,n4              ;number of bits
        jsr     <setvalue                   ;output the value

;bits 14 and 15 of MUSICAM frame header: mode extension
;               stereo intensity sub-band bound
;                       (applicable only to a joint stereo frame)

        move    y:stintns,y0                ;bits 14-15 of frame header
        move    #NSTINTENSITY,n4            ;number of bits
```

```
        jsr     <setvalue                   ;output the value

;bits 16 thru 19 of MUSICAM frame header:
;          16 = copyright: YES (1) or NO (0)
;          17 = original/home: copy (0) or original (1)
;       18-19 = emphasis

        clr     a                           ;to initialize bits 16 thru 19
        move    a,x:tstfrme                 ;temp variable to set the bits
        TST_CLR_HEADER_BIT_16_CD,_syst_50       ;if not set, continue
        bset    #3,x:tstfrme                ;set copyright bit

_syst_50
        TST_CLR_HEADER_BIT_17_CD,_syst_60       ;if not set, continue
        bset    #2,x:tstfrme                ;set original bit

_syst_60
        TST_CLR_HEADER_BIT_18_CD,_syst_70       ;if not set, continue
        bset    #1,x:tstfrme                ;set bit 1 of emphasis

_syst_70
        TST_CLR_HEADER_BIT_19_CD,_syst_80       ;if not set, continue
        bset    #0,x:tstfrme                ;set bit 0 of emphasis

_syst_80

;output frame header bits 16 thru 19

        move    x:tstfrme,y0                ;formatted bits
        move    #NSYSTHDR_3,n4              ;number of bits
        jsr     <setvalue                   ;output the value

        rts
```

```
        opt fc,mex
;
;  (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UXCODE\setdata.asm

        title   'Set the Data'

; This routine sets the data in the output buffer

; on entry
;       y:<usedsb = number of used sub-bands
;       r3 = address of left & right channel SubBandPosition array (x memory)
;       r2 = address of left & right channel SubBandSKFs array (x memory)
;       r1 = address of the left channel poly analyzed data
;       r0 = address of the right channel poly analyzed data
;       y:opfrtyp = whether full stereo, joint stereo or mon frame
;       y:<stereo = flags:
;                   bit 0 means stereo vs mono framing
;                               0 = stereo framing
;                               1 = mono framing
;                   bit 2 is to simply indicate that joint stereo applies
;                               0 = NOT joint stereo framing type
;                               1 = IS joint stereo framing type
;                   bit 3 is to indicate the full stereo upgrade by allocate rtn
;                       if joint stereo applies
;                               0 = normal joint stereo allocation
;                               1 = FULL STEREO allocation
;                   bit 4 is to simply indicate the stereo intensity sub-band
;                        boundary has been reached if joint stereo applies
;                               0 = NO sub-bands still below intensity boundary
;                               1 = sub-bands above intensity boundary
;       y:sibound = if joint stereo, sub-band boundary for stereo intensity


; on exit
;       a = destroyed
;       b = destroyed
;       x0 = destroyed
;       y0 = destroyed
;       x1 = destroyed
;       y1 = destroyed
;       r0 = destroyed
;       r2 = destroyed
;       r3 = destroyed
;       r4 = destroyed
;       r5 = destroyed
;       n5 = destroyed

        include 'def.asm'
        include 'box_ctl.asm'
        include '..\uxcode\quantize.mac'
        include '..\uxcode\setvalue.mac'

        section ytables
        xdef    NBits,AA,BB

        org     yhe:
stsetdata_yhe
```


BAD ORIGINAL

```
NBits
        dc      0                               ;position = 0, place holder
        dc      2                               ;position = 1
        dc      3                               ;position = 2
        dc      3                               ;position = 3
        dc      4                               ;position = 4
        dc      4                               ;position = 5
        dc      5                               ;position = 6
        dc      6                               ;position = 7
        dc      7                               ;position = 8
        dc      8                               ;position = 9
        dc      9                               ;position = 10
        dc      10                              ;position = 11
        dc      11                              ;position = 12
        dc      12                              ;position = 13
        dc      13                              ;position = 14
        dc      14                              ;position = 15
        dc      15                              ;position = 16
        dc      16                              ;position = 17

AA
        dc      $000000         ; position = 00, place holder
        dc      $600000         ; position = 01, .750000000
        dc      $500000         ; position = 02, .625000000
        dc      $700000         ; position = 03, .875000000
        dc      $480000         ; position = 04, .562500000
        dc      $780000         ; position = 05, .937500000
        dc      $7c0000         ; position = 06, .968750000
        dc      $7e0000         ; position = 07, .984375000
        dc      $7f0000         ; position = 08, .992187500
        dc      $7f8000         ; position = 09, .996093750
        dc      $7fc000         ; position = 10, .998046875
        dc      $7fe000         ; position = 11, .999023438
        dc      $7ff000         ; position = 12, .999511719
        dc      $7ff800         ; position = 13, .999755859
        dc      $7ffc00         ; position = 14, .999877930
        dc      $7ffe00         ; position = 15, .999938965
        dc      $7fff00         ; position = 16, .999969482
        dc      $7fff80         ; position = 17, .999984741

BB
        dc      $000000         ; position = 00, place holder
        dc      $600000         ; position = 01, 1.0-.250000000
        dc      $500000         ; position = 02, 1.0-.375000000
        dc      $700000         ; position = 03, 1.0-.125000000
        dc      $480000         ; position = 04, 1.0-.437500000
        dc      $780000         ; position = 05, 1.0-.062500000
        dc      $7c0000         ; position = 06, 1.0-.031250000
        dc      $7e0000         ; position = 07, 1.0-.015625000
        dc      $7f0000         ; position = 08, 1.0-.007812500
        dc      $7f8000         ; position = 09, 1.0-.003906250
        dc      $7fc000         ; position = 10, 1.0-.001953125
        dc      $7fe000         ; position = 11, 1.0-.000976563
        dc      $7ff000         ; position = 12, 1.0-.000488281
        dc      $7ff800         ; position = 13, 1.0-.000244141
        dc      $7ffc00         ; position = 14, 1.0-.000122070
        dc      $7ffe00         ; position = 15, 1.0-.000061035
        dc      $7fff00         ; position = 16, 1.0-.000030518
        dc      $7fff80         ; position = 17, 1.0-.000015259
```

```
endsetdata_yhe
        endsec

        section highmisc
        xdef    sample1
        xdef    sample2
        xdef    sample3

        org     xhe:
stsetdata_xhe

sample1 ds      1               ;1st sample of a triplet
sample2 ds      1               ;2nd sample of a triplet
sample3 ds      1               ;3rd sample of a triplet

endsetdata_xhe
        endsec

        section highmisc
        xdef    blleft,blright,SKFaddr,POSaddr,bandcnt,block
        xdef    MaxiAdd,MaxiFact

        org     yhe:
stsetdata_yhe

blleft          ds      1               ;left channel poly analyzed data
blright         ds      1               ;right channel poly analyzed data
SKFaddr         ds      1               ;save starting addr for SKF's
POSaddr         ds      1               ;save starting addr for SBIndx's
bandcnt         ds      1               ;incr sub-band for stereo intensity
block           ds      1               ;block no 0:0-3, 1:4-7, 2:8-11
MaxiAdd         ds      1               ;addr joint Maxi scale factors
MaxiFact        ds      1               ;joint Maxi scale factor

endsetdata_yhe
        endsec

;       org     pli:
        org     phe:

setdata
        move    r2,y:SKFaddr                    ;save start address
        move    r3,y:POSaddr                    ;save start address

        move    r1,y:blleft                     ;save left channel start addr
        move    r0,y:blright                    ;save right channel start addr

        move    #NUMSUBBANDS,n1                 ;spaced by number of subbands

        move    #0,r0                           ;start group number

;loop through the 12 groups of 3 samples per sub-band per channel
; advancing through 36 samples

        do      #NUMPERSUBBAND,_setd_90

;set which block of SKFs (scale factor indices):
;       0 for group of 4 samples 0-3
;       1 for group of 4 samples 4-7
;       2 for group of 4 samples 8-11
```

```
        move    r0,x0                         ;curr group to test
        move    #>4,b
        cmp     x0,b     #>0,y1               ;block [0] groups 0 - 3
        jgt     <_setd_00

        move    #>8,b
        cmp     x0,b     #>1,y1               ;block [1] groups 4 - 7
        jgt     <_setd_00

        move    #>2,y1                        ;block [2] groups 8 - 11

_setd_00
        move    (r0)+                         ;increment the group number
        move    y1,y:block                    ;save which block[0, 1 or 2]

;set-up for joint stereo channel sub-band intensity control

        move    #JntSBMaxi,r4                 ;addr of Joint Maxi factors
        move    y:sibound,n4                  ;joint stereo intensity sub-banc
        move    n4,y:bandcnt                  ;bound subband decremented cntr
        rep     #NPERGROUP
        move    (r4)+n4                       ;up JntMaxi table addr block 0
        move    r4,y:MaxiAdd                  ;save start of Joint Maxi facts
        bclr    #JOINT_at_SB_BOUND,y:<stereo  ;clear reached boundary sub-banc

;process for the defined used sub-bands this collection
; of three samples per sub-band per channel

        do      #NUMSUBBANDS,_setd_80

; if joint stereo does NOT apply, continue

        jclr    #JOINT_FRAMING,y:<stereo,_setd_08

; if joint stereo upgraded to full, continue

        jset    #JOINT_at_FULL,y:<stereo,_setd_08

; if doing joint stereo and have already switched over to joint SBits array,
;       continue by getting the Maxi factor for the block

        jset    #JOINT_at_SB_BOUND,y:<stereo,_setd_05

; see if the joint stereo intensity sub-boundary has been reached

        move    r3,y:svereg                   ;save reg 3
        move    y:bandcnt,r3                  ;get decrement sub-band ctr
        jsr     chkjoint                      ;see if reached boundary
        move    r3,y:bandcnt                  ;save new decremented ctr
        move    y:svereg,r3                   ;restore reg 3

; if intensity sub-band boundary NOT yet reached, continue

        jclr    #JOINT_at_SB_BOUND,y:<stereo,_setd_08

_setd_05

;get the Joint sub-band maxi factor for the group
```

```
        move    r3,y:svereg                          ;save reg 3
        move    y:MaxiAdd,r3                         ;get current Maxi sub-band
        move    y:block,n3                           ;which block for Maxi factor
        nop
        move    x:(r3+n3),y0                         ;get the maxi factor
        move    y0,y:MaxiFact                        ;save for quantize routine
        move    #NPERGROUP,n3                        ;position to next sub-band
        nop
        move    (r3)+n3                              ;adjust Maxi array addr to next
        move    r3,y:MaxiAdd                         ;save addr for next subband
        move    y:svereg,r3                          ;restore reg 3

_setd_08
        move    y:blleft,r1                          ;left channel block 1st
        move    #0,n3                                ;left channel SBIndx values
        move    y:block,n2                           ;which block of SKFs

;process left channel and then right channel for current sub-band

        do      #NUMCHANNELS,_setd_75

;now, if doing the left channel, continue with outputing data
;otherwise, check for joint stereo and the intensity bound of sub-band
;if right channel joint stereo sub-band intensity boundary reached,
;   skip putting out the right channel value for this sub-band
;otherwise output the true right channel stereo values to the frame

        jclr    #JOINT_at_SB_BOUND,y:<stereo,_setd_10 ;not joint boundary, go on
        move    n3,b                                 ;n3 is zero for left channel
        tst     b                                    ;test if doing left channel
        jne     _setd_70                             ;skip the right chan

_setd_10
        move    #BB,r4                               ;address of the B table
        move    x:(r3+n3),n5                         ;SubBandPosition[SubBand]
        move    n5,a                                 ;to test for NO index (0)
        tst     a           n5,n4                    ;check position == 0 AND
                                                     ; set position for BValue fetch
        jeq     _setd_70                             ;none to output, try next chan

        move    #AA,r5                               ;address of the A table
        move    y:(r4+n4),x1                         ;BValue
        move    y:(r5+n5),x0                         ;AValue

        move    #NBits,r5                            ;address of NBits array
        move    #>1,y0                               ;test type of group
        move    y:(r5+n5),n4                         ;nbits

        move    x:(r2+n2),n5                         ;SKFIndex[SubBand][block]
        move    #IvSKF,r5                            ;IvSKF table address

; test the position and pack those that qualify

        cmp     y0,a      #>2,y0                     ;check position == 1
        jeq     <_setd_30
        cmp     y0,a      #>4,y0                     ;check position == 2
        jeq     <_setd_40
        cmp     y0,a      #>3,y0                     ;check position == 4
        jeq     _setd_50
        cmp     y0,a                                 ;check pos == 3, and if not
```

```
        jne     <_setd_15                          ;handle all others not packed

; if not compressed mode, handle allocation position 3 normally
; if compression applies and NCT at the HIGH sampling rate,
;   handle allocation position 3 as a packed value

        jset    #USE_COMPRESS y:<cmprsctl,_setd_45

; not position 1, 2, (3, if compression) or 4;
;   just a regular output of 3 adjacent data values

_setd_15
        do      #NPERGROUP,_setd_20
        move    x:(r1)+n1,y0                       ;get data value
;       jsr     quantize                           ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        move    a1,y0                              ;move result into right reg
        clr     a               n4,b               ;set up a register for setvalue
                                                   ; & set len for setvalue macro
        move    y0,a0                              ;set up for setvalue macro
;       jsr     setvalue                           ;output the value
;MACRO: output the value
        SETVALUE
        nop

_setd_20
        jmp     _setd_70

; Pos 1: Three adjacent data values are packed into 5 bits.
;        Each of the data values are only 2 bits wide.
;
;       packed_value = value0 * 9 + value1 * 3 + value2
;                          or
;       packed_value = 3 * (value0 * 3 + value1) + value2

_setd_30
        move    x:(r1)+n1,y0                       ;get 1st data value
        move    y0,x:sample1
        move    x:(r1)+n1,y0                       ;get 2nd data value
        move    y0,x:sample2
        move    x:(r1)+n1,y0                       ;get 3rd data value

;if new ISO CRC, also code CCS corrction to packed values
;   which switches the 1st and 3rd values in the triplet
;   for ISO, 3rd value is correctly in place already in a register
;   for CCS, save sample 3 and retrieve 1st sample into a register

        jset    #CRC_OLD_vs_NEW,y:<stereo,_setd_31
        move    y0,x:sample3
        move    x:sample1,y0

_setd_31
;       jsr     quantize                           ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        move    a1,b                               ;set to mult value by 3
        lsl     b       #0,a0                      ;by 2
                                                   ; & kill extra bits
        add     a,b     x:sample2,y0               ;add for by 3 saving result in b
```

```
                                                  ; & get 2nd data value
                                                  ;quantize the data
;       jsr     quantize
;MACRO: quantize the data
        QUANTIZE
        move    #0,a0                             ;kill extra bits
        add     a,b                               ;add 2nd to mult value by 3
        lsl     b           b,a                   ;by 2
                                                  ; & save total to add for by 3

;if new ISO CRC, also code CCS correction to packed values
;   which switches the 1st and 3rd values in the triplet
;   for ISO, 1st value is correctly in place already in a register
;   for CCS, retrieve 3rd sample into a register

        add     a,b         x:sample1,y0          ;add for by 3 saving result in b
                                                  ; & set sample 1 as 3rd sample
        jset    #CRC_OLD_vs_NEW,y:<stereo,_setd_32
        move    x:sample3,y0                      ;set 3rd sample

_setd_32                                          ;quantize the data
;       jsr     quantize
;MACRO: quantize the data
        QUANTIZE

        add     b,a         #5,n4                 ;add in last result
                                                  ; & nbits result for setvalue
        move    a1,y0                             ;move to right register

;if compression applies:
; a. switch the bit count for setvalue
; b. set value for compression as register offset
; c. get the compressed value for setvalue

        jclr    #USE_COMPRESS,y:<cmprsctl,_setd_33
        move    #4,n4                             ;compress nbits for setvalue
        move    a1,n0                             ;move to right register
        jsr     compval                           ;get compressed value

_setd_33
        clr     a                   n4,b          ;set up a register for setvalue
                                                  ; & set len for setvalue macro
        move    y0,a0                             ;set up for setvalue macro
;       jsr     setvalue                          ;output the value
;MACRO: output the value
        SETVALUE
        jmp     _setd_70

; Pos 2: Three adjacent data values are packed into 7 bits.
;        Each of the data values are only 3 bits wide.
;
;       packed_value = value0 * 25 + value1 * 5 + value2
;                          or
;       packed_value = 5 * (value0 * 5 + value1) + value2

_setd_40
        move    x:(r1)+n1,y0                      ;get 1st data value
        move    y0,x:sample1
        move    x:(r1)+n1,y0                      ;get 2nd data value
        move    y0,x:sample2
        move    x:(r1)+n1,y0                      ;get 3rd data value
```

```
;if new ISO CRC, also code CCS corrction to packed values
;   which switches the 1st and 3rd values in the triplet
;   for ISO, 3rd value is correctly in place already in a register
;   for CCS, save sample 3 and retrieve 1st sample into a register

        jset    #CRC_OLD_vs_NEW,y:<stereo,_setd_41
        move    y0,x:sample3
        move    x:sample1,y0

_setd_41
;       jsr     quantize                        ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        move    a1,b                            ;set to mult value by 5
        lsl     b       #0,a0                   ;by 2
                                                ; & kill extra bits
        lsl     b                               ; by 4 (2 again)
        add     a,b     x:sample2,y0            ;add for by 5 saving result in b
                                                ; & get 2nd data value
;       jsr     quantize                        ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        move    #0,a0                           ;kill extra bits
        add     a,b                             ;add 2nd to mult value by 5
        lsl     b       b,a                     ;by 2
                                                ; & save total to add for by 5
        lsl     b                               ;by 4 (2 again)

;if new ISO CRC, also code CCS corrction to packed values
;   which switches the 1st and 3rd values in the triplet
;   for ISO, 1st value is correctly in place already in a register
;   for CCS, retrieve 3rd sample into a register

        add     a,b     x:sample1,y0            ;add for by 5 saving result in b
                                                ; & set sample 1 as 3rd sample
        jset    #CRC_OLD_vs_NEW,y:<stereo,_setd_42
        move    x:sample3,y0

_setd_42
;       jsr     quantize                        ;quantize the data
;MACRO: quantize the data
        QUANTIZE

        add     b,a     #7,n4                   ;add in last result
                                                ; & nbits result for setvalue
        move    a1,y0                           ;move to right register

;if compression applies:
; a. switch the bit count for setvalue
; b. set value for compression as register offset
; c. get the compressed value for setvalue

        jclr    #USE_COMPRESS,y:<cmprsctl,_setd_43
        move    #6,n4                           ;compress nbits for setvalue
        move    a1,n0                           ;move to right register
        jsr     compval                         ;get compressed value

_setd_43
        clr     a               n4,b            ;set up a register for setvalue
```

```
                                                  ; & set len for setvalue macro
        move    y0,a0                             ;set up for setvalue macro
;       jsr     setvalue                          ;output the value
;MACRO: output the value
        SETVALUE
        jmp     _setd_70

_setd_45

;if compression applies for position 3:
; Pos 3: Three adjacent data values are packed into 8 bits.
;        Each of the data values are only 3 bits wide.
;
;       packed_value = value0 * 64 + value1 * 8 + value2
;                               or
;       packed_value = 8 * (value0 * 8 + value1) + value2

        move    x:(r1)+n1,y0                      ;get 1st data value
;       jsr     quantize                          ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        move    a1,b                              ;set to mult value by 8
        lsl     b                                 ;by 2
        lsl     b                                 ;by 4 (2 again)
        lsl     b       x:(r1)+n1,y0              ;by 8 (2 again) save result in b
                                                  ; & get 2nd value
;       jsr     quantize                          ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        move    #0,a0                             ;kill extra bits
        add     a,b                               ;add to total to mult value by 8
        lsl     b                                 ;by 2
        lsl     b                                 ;by 4 (2 again)
        lsl     b       x:(r1)+n1,y0              ;by 8 (2 again) save result in b
                                                  ; & get 3rd value
;       jsr     quantize                          ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        add     b,a     #8,n4                     ;add in last result
                                                  ; & nbits result for setvalue
        move    a1,n0                             ;move to right register
        jsr     compval                           ;get compressed value
        clr     a               n4,b              ;set up a register for setvalue
                                                  ; & set len for setvalue macro
        move    y0,a0                             ;set up for setvalue macro
;       jsr     setvalue                          ;output the value
;MACRO: output the value
        SETVALUE
        jmp     _setd_70

; Pos 4: Three adjacent data values are packed into 10 bits.
;        Each of the data values are only 4 bits wide.
;
;       packed_value = value0 * 81 + value1 * 9 + value2
;                               or
;       packed_value = 9 * (value0 * 9 + value1) + value2

_setd_50
        move    x:(r1)+n1,y0                      ;get 1st data value
        move    y0,x:sample1
```

```
        move    x:(r1)+n1,y0                    ;get 2nd data value
        move    y0,x:sample2
        move    x:(r1)+n1,y0                    ;get 3rd data value

;if new ISO CRC, also code CCS corrction to packed values
;   which switches the 1st and 3rd values in the triplet
;   for ISO, 3rd value is correctly in place already in a register
;   for CCS, save sample 3 and retrieve 1st sample into a register

        jset    #CRC_OLD_vs_NEW,y:<stereo,_setd_51
        move    y0,x:sample3
        move    x:sample1,y0

_setd_51
;       jsr     quantize                        ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        move    a1,b                            ;set to mult value by 9
        lsl     b       #0,a0                   ;by 2
                                                ; & kill extra bits
        lsl     b                               ;by 4 (2 again)
        lsl     b                               ;by 8 (2 again)
        add     a,b     x:sample2,y0            ;add for by 9 saving result in b
                                                ; & get 2nd value
;       jsr     quantize                        ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        move    #0,a0                           ;kill extra bits
        add     a,b                             ;add 2nd to mult value by 9
        lsl     b       b,a                     ;by 2
                                                ; & save total to add for by 9
        lsl     b                               ;by 4 (2 again)
        lsl     b                               ;by 8 (2 again)

;if new ISO CRC, also code CCS corrction to packed values
;   which switches the 1st and 3rd values in the triplet
;   for ISO, 1st value is correctly in place already in a register
;   for CCS, retrieve 3rd sample into a register

        add     a,b     x:sample1,y0            ;add for by 9 saving result in b
                                                ; & set sample 1 as 3rd sample
        jset    #CRC_OLD_vs_NEW,y:<stereo,_setd_52
        move    x:sample3,y0

_setd_52
;       jsr     quantize                        ;quantize the data
;MACRO: quantize the data
        QUANTIZE
        add     b,a     #10,n4                  ;add in last result
                                                ; & nbits result for setvalue
        move    a1,y0                           ;move to right register
        clr     a               n4,b            ;set up a register for setvalue
                                                ; & set len for setvalue macro
        move    y0,a0                           ;set up for setvalue macro
;       jsr     setvalue                        ;output the value
;MACRO: output the value
        SETVALUE

; We have just finished the current channel
;  and since the left was 1st, set up for the right channel
```


BAD ORIGINAL

```
_setd_70
        move    y:blright,r1                    ;now right channel block
        move    #>NUMSUBBANDS*NPERGROUP,a       ;move to SKFs for right channel
        move    y:block,x0                      ;get current block offset
        add     x0,a    #NUMSUBBANDS,n3         ;add right chan offset, set
                                                ; AND set adj to right SBPos
        move    a1,n2                           ;offset register 2

; We have just finished both channels for a sub-band.
;   1. adjust left and right poly analyzed sample pointers to next sub-band
;   2. increment SBPos array pointer for next sub-band
;   3. increment the SKFs array pointer over previous sub-band's 2nd & 3rd SKFs

_setd_75
        move    #>1,x0                          ;incr left and right rcv'd samps
        move    y:blleft,a                      ;left address prev sub-band
        add     x0,a    y:blright,b             ;adj left chan, get right chan
        move    a,y:blleft                      ;save left addr next sub-band
        add     x0,b    (r3)+                   ;adj right chan, incr SBPos ptr
        move    #3,n2                           ;adj SKFs by 3
        move    b,y:blright                     ;save right addr next sub-band
        move    (r2)+n2                         ;next sub-band SKFs addr

_setd_80

;We have just finished a group of 3 samples per sub-band a'd we must
;  get set for the next group or 3 samples:
; 1. adjust the left and right poly analyzed sample pointers for
;    the 2nd and 3rd samples in the group just finished
; 2. restore the starting address of the SBPos array
; 3. restore the starting address of the SKFs array
; 4. restore joint stereo sub-band intensity boundary

        move    #>NUMSUBBANDS*2,x0              ;adj over 2nd & 3rd samples
        move    y:blleft,a                      ;left address prev sub-band
        add     x0,a    y:blright,b             ;adj left ptr, get right ptr
        move    a,y:blleft                      ;save left addr next group
        add     x0,b    y:POSaddr,r3            ;adj right ptr, reset SBPos ptr
        move    b,y:blright                     ;save right addr next group
        move    y:SKFaddr,r2                    ;reset start SKF address

_setd_90
        rts
```

```
        opt     fc
;
;  c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\tstsine.asm
        title   'Check Maskers for Sine'

; This is a routine to test the Tonals for the presence of a sine wave.

;on entry: r0 == addr of the Maskers structure array

        include 'def.asm'

        org     phe:
tstsine
        move    r0,x:<SvReg0            ;save addr of Maskers array

;set the frame counter and sine flag from the proper channel

        move    y:sincntlft,x0          ;start with the left channel
        move    y:sintstlft,x1          ;start with left channel test flag
        jclr    #LEFT_vs_RIGHT,y:<stereo,_tsin_00  ;if left, continue
        move    y:sincntrgt,x0          ;switch to the right channel
        move    y:sintstrgt,x1          ;switch to right channel test flag

_tsin_00

;set the working variables with the values for the proper channel

        move    x0,x:<sincnt            ;set the working frame count value
        move    x1,x:<sintest           ;set sine flag for proper channel

;start looking for a sine wave in the current channel

        move    #>MINDB,x0              ;minimum value
        move    x0,x:<maxtonal          ;set minimum value for max tonal
        move    x:<nmasker,b            ;number of maskers in array
        move    #>TONAL,x1              ;to match TONAL only 1st pass

;loop thru the maskers array looking for the highest tonal

        do      b,_tsin_20
        move    #MASKERSTYPE,n0         ;offset for type of masker
        nop
        move    x:(r0+n0),a             ;get curr masker's type
        cmp     x1,a    x:<maxtonal,y0  ;check if it's a tonal
                                        ; & get set to compare to curr max
        jne     _tsin_10                ;if not a TONAL, continue

;test the power vs last high tonal power

        move    #MASKERSPWRDB,n0        ;offset to PowerDb
        nop
        move    x:(r0+n0),a             ;get TONAL PowerDb
        cmp     y0,a    #MASKERSBIN,n0  ;compare curr to last max TONAL value
                                        ; & get set to save bin # if higher
        jle     _tsin_10                ;not a new higher PowerDB, continue

        move    a,x:<maxtonal           ;save new max tonal
```


BAD ORIGINAL

```
        move    x:(r0+n0),y0           ;get the bin number
        move    y0,x:<maxbin

_tsin_10
        move    #MASKERSSIZE,n0        ;size of Masker structure
        nop
        move    (r0)+n0                ;advance to next Masker structure

_tsin_20

;now that we have the max tonal, test if another masker is within 30 Db

        move    #30/192.66,x0          ;subtract 30 Db from max tonal
        move    x:<maxtonal,a          ;get the max tonal PowerDb
        sub     x0,a    #>-1,x1        ;subtract off 30 Db
                                       ; & set 2nd sub-band NOT a sine to XCODE
        move    a,y1                   ;value to check against
        move    #>-1,x0                ;set 1st sub-band NOT a sine to XCODE
        move    x:<SvReg0,r0           ;address the Masker structure

;loop thru the maskers array looking for the highest tonal

        do      b,_tsin_40
        move    #MASKERSBIN,n0         ;offset to bin number
        move    x:<maxbin,y0           ;to see if this is selected as max
        move    x:(r0+n0),a            ;get bin number
        cmp     y0,a    #MASKERSPWRDB,n0        ;check if selected as max
                                       ; & set offset to PowerDb
        jeq     _tsin_30               ;it's the selected one, continue

;test the power vs last high tonal power

        move    x:(r0+n0),a            ;get masker PowerDb
        cmp     y1,a                   ;compare curr to max TONAL - 30 Db
        jle     _tsin_30               ;not a new higher PowerDB, continue

;if PowerDb is within 30 Db, it's NOT a sine wave, stop checking

        enddo
        jmp     _tsin_100

_tsin_30
        move    #MASKERSSIZE,n0        ;size of Masker structure
        nop
        move    (r0)+n0                ;advance to next Masker structure

_tsin_40

;to test consecutive frame count before declaring a sine wave in a channel

        move    #>SINE_FRAME_COUNT,y0

; set channel as a sine wave after ensuring the sine wave persists

        move    x:<sincnt,a
        cmp     y0,a    #>1,y0
        jge     _tsin_50

;count another frame set as a sine wave
```

```
        add     y0,a
        move    a,x:<sincnt
        jmp     _tsin_900

_tsin_50

;now set channel as a sine wave

        bset    #LEFT_SINE_WAVE,x:<sintest

;we have a sine wave, determine the two sub-bands with the sine wave

        move    x:<maxbin,b             ;get the bin number and divide by 16
        asr     b                       ;divide by 2
        asr     b                       ;divide by 4
        asr     b                       ;divide by 8
        asr     b                       ;divide by 16
        move    b,r0                    ;save the sub-band

;now see if this is the 1st sub-band to increment for 2nd sub-band
;  OR is this the 2nd sub-band to decrement for 1st sub-band
;mask off all but the lower 4 bits of bin number

        move    x:<maxbin,b             ;get the bin number
        move    #>$F,x0                 ;to mask off all but lower 4 bits
        and     x0,b     #>8,x0         ;mask off bits
                                        ; & set to test for increment
        cmp     x0,b                    ;if greater, increment for 2nd sub-band
        jgt     _tsin_70

        move    r0,x1                   ;this is the 2nd sub-band of the pair
        move    r0,b                    ;check if sub-band 0
        tst     b                       ;check for sub-band 0
        jeq     _tsin_60                ;if 0, 1st sub-band equals 2nd sub-band
        move    (r0)-                   ;set 1st sub-band as previous

_tsin_60
        move    r0,x0                   ;insert the 1st sub-band of the pair
        jmp     _tsin_900

_tsin_70
        move    r0,x0                   ;this is the 1st sub-band of the pair
        move    (r0)+                   ;set 2nd sub-band as next sequential
        move    r0,x1
        jmp     _tsin_900

_tsin_100

;determined as NOT a sine wave, see if previously set as a sine wave
; if channel was not defined as a sine wave, DONE!!

        jclr    #LEFT_SINE_WAVE,x:<sintest,_tsin_900

;set consecutive count before declaring a sine wave in a channel

        move    #>SINE_FRAME_COUNT,y0

;see that the sine wave has stopped persisting for N frames

        move    x:<sincnt,a
```

```
        tst     a           #>1,y0
        jeq     _tsin_110

;decrement another frame NOT as a sine wave

        sub     y0,a
        move    a,x:<sincnt

;restore previous found sub-bands

        jset    #LEFT_vs_RIGHT,y:<stereo,_tsin_105

;reset for the left channel of the pair

        move    y:strtsinlft,x0         ;left channel last found 1st sub-band
        move    y:endsinlft,x1          ;left channel last found 2nd sub-band

        jmp     _tsin_900               ;DONE!!!

_tsin_105

;reset for the right channel of the pair

        move    y:strtsinrgt,x0         ;right channel last found 1st sub-band
        move    y:endsinrgt,x1          ;right channel last found 2nd sub-band

        jmp     _tsin_900               ;DONE!!!

_tsin_110

;now clear the channel as a sine wave

        bclr    #LEFT_SINE_WAVE,x:<sintest
;       jmp     _tsin_900               ;DONE!!!

_tsin_900

        rts
```


BAD ORIGINAL

```
        opt fc,mex,cex
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\trapcell.asm

        title   'Trap Cells'

;  xcode dsp trapcell.asm

        section trapcell

        org     p:$0
        jmp     >start

; IRQA:
;       react to the frame time millisecond interval
;         qtalloc interrupt (quit bit allocation) for bit allocation

        org     p:$8
        jsr     >irqa

; IRQB:
;       react to the frame time millisecond interval
;         timer interrupt (start XPSYCHO and XCODE of new frame)

        org     p:$a
        jsr     >irqb

;SSI receive data interrupt:
;         copy in next input PCM value from A-to-D converter

        org     pli:$c
        jsr     <ssirec
        nop

;SSI receive data interrupt with exceptions:
;         copy in next input PCM value from A-to-D converter

        org     pli:$e
        jsr     <ssirece                   ;handle input channel pcm data exception
        nop

; SSI transmit data interrupt:
;         output the next encoded frame word from buffer

        org     p:$10
        jsr     <ssixmt
        nop

; SSI transmit data interrupt with exceptions:
;         output the next encoded frame word from buffer

        org     p:$12
        jsr     <ssixmte
        nop

; SCI receive serial communications interrupt:
;       input the next ancillary data byte
```

```
        org     p:$14
        jsr     <scirec
        nop

; SCI receive serial data interrupt with exceptions:
;       input the next ancillary data byte

        org     p:$16
        jsr     <scirece
        nop

; HOST COMMAND - 24: get the encoder switches host vector

        org     p:$24
        jsr     >hostvector_24

; HOST COMMAND - 26: get the encoder framing type host vector

        org     p:$26
        jsr     >hostvector_26

; HOST COMMAND - 28: get the encoder iso header host vector

        org     p:$28
        jsr     >hostvector_28

; HOST COMMAND - 2A: get the psycho table offset ID for a new parameter value

        org     p:$2a
        jsr     >hostvector_2A

; HOST COMMAND - 2C: update the psycho table with a new parameter value

        org     p:$2c
        jsr     >hostvector_2C

; HOST COMMAND - 2E: clean host vector buffer: read double buffer

        org     p:$2e
        jsr     >hostvector_2E

; HOST COMMAND - 30: indicate to the host that the encoder interrupts
;                       are on and functioning

        org     p:$30
        jsr     >hostvector_30

;unexpected interrupts

        org     p:$2
        jsr     >stack_error
        org     p:$1a
        jsr     >sciidle_line
        org     p:$1c
        jsr     >scitimer
        org     p:$3e
        jsr     >illegal_inst

        endsec
```

```
        opt fc
;
;  c! 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\ssirec.asm

        title 'SSI receive data interrupt handler'

        include 'def.asm'
        include 'box_ctl.asm'
        include '..\common\ioequ.asm'

;these save variables for exclusive use by the ssirec interrupt handlers only

        section lowmisc
        xdef    ssirecR7Save
        xdef    ssirecM7Save

        org     yli:
stssirec_yli

ssirecR7Save    ds      1
ssirecM7Save    ds      1

endssirec_yli
        endsec

; SSI Receiver interrupt

        org     pli:

ssirec
        move    r7,y:<ssirecR7Save      ;save register
        move    m7,y:<ssirecM7Save      ;save register

;set up to receive this next input PCM data value

        move    y:<ipwptr,r7            ;curr input PCM data write pointer
        move    #PCMSIZE*2-1,m7         ;set as a mod buffer for both channels

;!!!12/14/94
        nop
;test for which channel is incoming and align the pointer if needed
;if it;s a right channel value, capture it to current address in buffer
;if it's a left channel value,
;  left channel values are stored on even buffer addresses the right channel
;  is stored in the adjacent odd buffer address

;;      TST_SET_RIGHT_PCM_INPUT_XPS,_ssi_05     ;if low, its the right channel

;see if a left channel input PCM data buffer address realignment is needed

;;      jclr    #0,y:<ipwptr,_ssi_05    ;if addr already even, continue

;align odd buffer address to even for the left channel addresses
; NOTE: this alignment should occur only once during steady operation

;;      move    (r7)-                   ;align for left channel values

;;_ssi_05
```


BAD ORIGINAL

```
;!!!12/14/94

;capture the new input PCM value and store in the buffer (properly aligned)

        movep   x:<<M_RX,x:(r7)+        ;input the current channel PCM value
                                        ; & advance to next channel position
        move    r7,y:<ipwptr            ;save addr for the input PCM value

        move    y:<ssirecR7Save,r7      ;restore register
        move    y:<ssirecM7Save,m7      ;restore register
        rti

; SSI Receiver interrupt with exceptions

ssirece
        move    r7,y:<ssirecR7Save      ;save register
        movep   x:<<M_SR,r7             ;clear the exeption
        movep   x:<<M_RX,r7             ;eat the input the data
        move    y:<ssirecR7Save,r7      ;restore register

        rti
```

```
        opt fc
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\ssixmt.asm

        title 'SSI interrupt handler'

; xcode dsp ssixmt.asm

        include '..\common\ioequ.asm'

;these save variables for exclusive use by the ssixmt interrupt handlers only

        section lowmisc
        xdef    ssixmtR7Save
        xdef    ssixmtM7Save

        org     yli:
stssixmt_yli

ssixmtR7Save    ds      1
ssixmtM7Save    ds      1

endssixmt_yli
        endsec

; SSI Transmitter interrupt

        org     pli:

ssixmt
        move    r7,y:<ssixmtR7Save
        move    m7,y:<ssixmtM7Save

        move    y:<oprptr,r7            ;get output read buffer pointer
        move    y:<outsize,m7           ;circular buffer (2 frames worth)
        nop
        movep   y:(r7)+,x:<<M_TX        ;output word for the rdecode
        move    r7,y:<oprptr            ;update output read buffer pointer

        move    y:<ssixmtM7Save,m7
        move    y:<ssixmtR7Save,r7

        rti

; SSI Transmitter interrupt with exceptions

ssixmte
        move    r7,y:<ssixmtR7Save

        movep   x:<<M_SR,r7             ;clear the exeption
        movep   x:(r7)+,x:<<M_TX        ;output the data

        move    y:<ssixmtR7Save,r7

        rti
```

```
        opt     fc,cex
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\polyanal.asm

        title   'Analysis Polyphase Filter'

; This routine performs the polyphase analysis filter on an input data
; set of 32 samples.
; The input data is assumed to be ordered so the oldest data is at higher
; addresses. Newer data is put in at the "left" of the old data.
;
; Observe the following about the M's
;
; k = 0..63
;
;       M[00][k]  = M[31][k]              even k
;       .                                 .
;       .                                 .
;       .                                 .
;       M[14][k]  = M[17][k]              .
;       M[15][k]  = M[16][k]              .
;
;       M[00][k]  = -M[31][k]             odd k
;       .                                 .
;       .                                 .
;       .                                 .
;       M[14][k]  = -M[17][k]             .
;       M[15][k]  = -M[16][k]             .
;
; Thus the S's can be calculated with one half of the calculations as
; follows:
;
;       The origional formula
;
;       S(i) = sum(k=0..63) M(i,k) * Y(k)   i = 0..31
;
; Now using the symmetry of the M's
;
; The new way first calculates the following 32 values
;
; define YP[k] k=0..31 as follows
;       YP[ 0] = Y[ 0] + Y[32]
;       YP[ 2] = Y[ 2] + Y[30]
;       YP[ 4] = Y[ 4] + Y[28]
;       YP[ 6] = Y[ 6] + Y[26]
;       YP[ 8] = Y[ 8] - Y[24]
;       YP[10] = Y[10] + Y[22]
;       YP[12] = Y[12] - Y[20]
;       YP[14] = Y[14] - Y[18]
;       YP[16] = Y[16]
;       YP[18] = Y[34] - Y[62]
;       YP[20] = Y[36] - Y[60]
;       YP[22] = Y[38] - Y[58]
;       YP[24] = Y[40] - Y[56]
;       YP[26] = Y[42] - Y[54]
;       YP[28] = Y[44] - Y[52]
;       YP[30] = Y[46] - Y[50]
;
```

```
;        YP[ 1] = Y[ 1] - Y[31]
;        YP[ 3] = Y[ 3] - Y[29]
;        YP[ 5] = Y[ 5] - Y[27]
;        YP[ 7] = Y[ 7] - Y[25]
;        YP[ 9] = Y[ 9] - Y[23]
;        YP[11] = Y[11] - Y[21]
;        YP[13] = Y[13] - Y[19]
;        YP[15] = Y[15] - Y[17]
;        YP[17] = Y[33] - Y[63]
;        YP[19] = Y[35] - Y[61]
;        YP[21] = Y[37] - Y[59]
;        YP[23] = Y[39] - Y[57]
;        YP[25] = Y[41] - Y[55]
;        YP[27] = Y[43] - Y[53]
;        YP[29] = Y[45] - Y[51]
;        YP[31] = Y[47] - Y[49]
;
;        i = 0..15
;old way         even(i) = sum(k=0,2,4,..62) M(i,k) * Y(k)
;new way         even(i) =       M(i, 0)*YP( 0) +  M(i, 2)*YP( 2) +
;                                M(i, 4)*YP( 4) +  M(i, 6)*YP( 6) +
;                                M(i, 8)*YP( 8) +  M(i,10)*YP(10) +
;                                M(i,12)*YP(12) +  M(i,14)*YP(14) +
;                                M(i,16)*YP(16) +  M(i,34)*YP(18) +
;                                M(i,36)*YP(20) +  M(i,38)*YP(22) +
;                                M(i,40)*YP(24) +  M(i,42)*YP(26) +
;                                M(i,44)*YP(28) +  M(i,46)*YP(32)
;
;old way         odd(i) = sum(k=1,3,5,..63) M(i,k) * Y(k)
;new way         odd(i) =        M(i, 1)*YP( 1) + M(i, 3)*YP( 3) +
;                                M(i, 5)*YP( 5) + M(i, 7)*YP( 7) +
;                                M(i, 9)*YP( 9) + M(i,11)*YP(11) +
;                                M(i,13)*YP(13) + M(i,15)*YP(15) +
;                                M(i,33)*YP(17) + M(i,35)*YP(19) +
;                                M(i,37)*YP(21) + M(i,39)*YP(23) +
;                                M(i,41)*YP(25) + M(i,43)*YP(27) +
;                                M(i,45)*YP(29) + M(i,47)*YP(31)
;
;                S(i) =  even(i) + odd(i)
;                S(31-i) = even(i) - odd(i)
;
; Based on the above, the M array is stored in memory as follows:
;
;       M[00][0] M[00][2] M[00][4] .. M[00][32] M[00][1] M[00][3] .. M[00][31]
;       .
;       .
;       .
;       M[15][0] M[15][2] M[15][4] .. M[15][32] M[15][1] M[15][3] .. M[15][31]
;
;
; on entry
;
;       r0(x) = address of the oldest input 32 PCM samples (32)
;               newest data at higher address
;       m0 = set properly
;
;       m2 = 63 (mod 64 buffer)
;
;       r3 = address of the next location in X array to place new data
;       m3 = 511 (mod 512 buffer)
```

```
;       r5(x) = address of the S output vector (32)

;       !!! this routine leaves the m registers set like on entry

;       The X buffer must be allocated so it can be a mod buffer (512).
;       The Y buffer must be allocated so it can be a mod buffer (64).

; on exit
;       r0 = updated (incremented) for next interation.
;       r3 = updated (decremented) to beginning of x array for next interation
;       r5 = updated to point to input S vector address + 32
;
;       a = destroyed
;       x0 = destroyed
;       y0 = destroyed
;       r0 = destroyed
;       r1 = destroyed
;       r2 = destroyed
;       r4 = destroyed
;       n1 = destroyed
;       n2 = destroyed
;       n3 = destroyed
;       n4 = destroyed
;       n5 = destroyed
        include 'def.asm'

        section polyanac
        xdef    polyc

        org     yli:
stpolyc_yli

        include '..\x1psycho\polyc.asm'

endpolyc_yli
        endsec

        section polyanam
        xdef    polym

        org     yhe:
stpolym_yhe

        include '..\x1psycho\polym.asm'

endpolym_yhe
        endsec

        org     pli:

panalysi

;       First move the pcm data into the x vector.
;       Remember that the oldest pcm data is at the highest address.
;

        do      #31,_poly15
        move    x:(r0)+n0,x0
        move    x0,x:(r3)-
```


BAD ORIGINAL

```
_polyis
        move    x:(r0)-n0,x0
        move    x0,x:(r3)

; At this point, r3 should point the the first valid data in x.  This address
; is the newest information.  As r3 is incremented, it points to older
; data.

;       Now the data is in the proper place

;       Window all the X data by the C vector.
;
;               Z(i) = C(i) * X(i)          i=0..511
;               C = r4
;               X = r3

; compute the Y vector
;               Y(i) = sum(j=0..7) Z(i+64j)        i = 0..63

;               Y = r2
;
; This version makes the observation that the Z vector is a temporary
; and thus Y can be computed as follows:
;
;               Y(i) = sum(j=0..7)[C(i+64j) * X(i+64j)]      i = 0..63
;
; This saves the storage space for Z and the store and load associated with Z.
;
; There is something curious about the C's. They possess a certian symmetry.
; The C's range from 0..511.  If one thinks about a new quantity called E, where
; where the E's are defined
;
;       E[000] = C[000]
;       E[001] = C[064]
;       .
;       E[007] = C[448]
;
;       E[008] = C[001]
;       E[009] = C[065]
;       .
;       E[015] = C[449]
;
;       E[504] = C[063]
;       E[505] = C[127]
;       . .
;       E[511] = C[511]

; Now observe that
;
;       E[259-i] = -E[260+i]            for i = 0..251
;
; This fact allows us to only store 256+16 of the E's (In fact if we were
; really clever with the code, we should only have to store 256 + 12 E's).
; The polyc array is really the E values.
; The trick is to try to store as much of the polyc array in low memory
; (0..ff) as possible so the parallel move proceeds as fast as possible
; for the mac instructions.

        move    #polyc,r4                       ;get addr of C window
        move    #ybuf,r2                        ;set address of y buf
```


BAD ORIGINAL

```
        move    #64,n3                                          ;set skip factor
        move    #9,n4                                           ;set to skip back

        do      #32,_poly20
        clr     a           x:(r3)-n3,x0    y:(r4)-,y0          ;get first data
        mac     x0,y0,a     x:(r3)-n3,x0    y:(r4)-,y0          ;compute Z
        mac     x0,y0,a     x:(r3)-n3,x0    y:(r4)-,y0          ;compute Z
        mac     x0,y0,a     x:(r3)-n3,x0    y:(r4)-,y0          ;compute Z
        mac     x0,y0,a     x:(r3)-n3,x0    y:(r4)-,y0          ;compute Z
        mac     x0,y0,a     x:(r3)-n3,x0    y:(r4)-,y0          ;compute Z
        mac     x0,y0,a     x:(r3)-n3,x0    y:(r4)-,y0          ;compute Z
        mac     x0,y0,a     x:(r3)-n3,x0    y:(r4)-,y0          ;compute Z
        mac     x0,y0,a     x:(r3)-n3,x0                        ;compute Z
        macr    x0,y0,a     (r3)-                               ; & position X for next
                                                                ;save as new Y
        move    a,x:(r2)-
_poly20
                                                                ;start and end
        move    (r4)-n4
        do      #31,_poly25
        clr     a           x:(r3)+n3,x0    y:(r4)-,y0          ;get first data
        mac     -x0,y0,a    x:(r3)+n3,x0    y:(r4)-,y0          ;compute Z
        mac     -x0,y0,a    x:(r3)+n3,x0    y:(r4)-,y0          ;compute Z
        mac     -x0,y0,a    x:(r3)+n3,x0    y:(r4)-,y0          ;compute Z
        mac     -x0,y0,a    x:(r3)+n3,x0    y:(r4)-,y0          ;compute Z
        mac     -x0,y0,a    x:(r3)+n3,x0    y:(r4)-,y0          ;compute Z
        mac     -x0,y0,a    x:(r3)+n3,x0    y:(r4)-,y0          ;compute Z
        mac     -x0,y0,a    x:(r3)+n3,x0    y:(r4)-,y0          ;compute Z
        mac     -x0,y0,a                                        ;compute Z
        macr    -x0,y0,a    (r3)+                               ; & position X for next
                                                                ;save as new Y
        move    a,x:(r2)-
_poly25                                                         ;adjust for next round
        move    (r3)-
        move    (r3)-n3

; The (r3)- and (r3)-n3 above is used to position r3 to the next  empty)
; position.  This position is one before the beginning of the array.
; This is at a lower addr.  This is the address for the NEXT new information.

; Lastly calculate the sub-band output (32 sub-bands)
;
;       i = 0..15
;               even(i) = see above
;               odd(i) = see above
;               S(i) =  even(i) + odd(i)
;               S(31-i) = even(i) - odd(i)
;
;               S = r5,r1
;               M = r4
;               Y = r2
;               a = even(i) sum
;               b = odd(i) sum

; First calculate the YP array from the Y array.

        move    #32,n4
        move    r2,r4                           ;set start address of YP array
        move    r2,r1
        move    (r4)-n4
```

```
;         r1 now points to YP[0]
;         r2 now points to Y[0]
;         r4 now points to Y[32]
;
          move     #2,n2
          move     n2,n4                       ;set output buffer increment
          move     n2,n1

          move     x:(r4)-n4,x0
          do #7,_poly26
          move     x:(r2)+n2,a
          add      x0,a     x:(r4)-n4,x0
          move     a,x:(r1)+n1
_poly26
          move     x:(r2)+n2,a                 ;now do the last one
          add      x0,a #18,n2
          move     a,x:(r1)-n1

;         Now r1 points to YP[16]
;         Now r2 points to Y[16]
;         Now r4 points to Y[16]

          move     (r1)+n1                     ;set r1 to point to YP[18]

          move     #46,n4
          move     (r2)+n2
          move     (r4)+n4

;         Now r1 points to YP[18]
;         Now r2 points to Y[34]
;         Now r4 points to Y[62]

          move     #2,n4
          move     n4,n2

          move     x:(r4)-n4,x0                ;now do YP[18]..YP[30] (even)
          do #6,_poly27
          move     x:(r2)+n2,a
          sub      x0,a     x:(r4)-n4,x0
          move     a,x:(r1)+n1
_poly27
          move     x:(r2)+n2,a
          sub      x0,a     #47,n2
          move     a,x:(r1)+n1

;         Now r1 points to YP[18]
;         Now r2 points to Y[48]
;         Now r4 points to Y[48]

          move     #17,n4
          move     (r2)-n2
          move     (r4)-n4

          move     r2,r1                       ;set to YP[1]

;         Now r1 points to YP[1]
;         Now r2 points to Y[1]
;         Now r4 points to Y[31]

          move     #2,n4
```

```
        move    n4,n2

        move    x:(r4)-n4,x0
        do #7,_poly28                           ;now do YP[15]..YP[31] (odd)
        move    x:(r2)-n2,a
        add     x0,a x:(r4)-n4,x0
        move    a,x:(r1)-n1
_poly28
        move    x:(r2)-n2,a
        add     x0,a #16,n2
        move    a,x:(r1)-n1

;       Now r1 points to YP[17]
;       Now r2 points to Y[17]
;       Now r4 points to Y[15]

        move    #48,n4
        move    (r2)+n2
        move    (r4)-n4

;       Now r1 points to YP[17]
;       Now r2 points to Y[33]
;       Now r4 points to Y[63]

        move    #2,n4
        move    n4,n2

        move    x:(r4)-n4,x0
        do #7,_poly29                           ;now do YP[17]..YP[31] (odd)
        move    x:(r2)+n2,a
        sub     x0,a x:(r4)-n4,x0
        move    a,x:(r1)+n1
_poly29
        move    x:(r2)+n2,a
        sub     x0,a    #polym,r4
        move    a,x:(r1)+n1

; Now we have the YP array all set

        move    #31,m2
        move    r5,r1                                   ;save start S addr
        move    #31,n1
        move    #ybuf,r2                                ;set start of YP buffer
        move    (r1)+n1                                 ;set to last addr

        do      #16,_poly30

; do even sums

        clr     a         x:(r2)-n2,x0     y:(r4)+,y0
;       rep     #15
        mac     x0,y0,a   x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,a   x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,a   x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,a   x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,a   x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,a   x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,a   x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,a   x:(r2)-n2,x0     y:(r4)+,y0
```

```
        mac     x0,y0,a x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,a x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,a x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,a x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,a x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,a x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,a x:(r2)-n2,x0     y:(r4)-,y0
        macr    x0,y0,a (r2)-

; do odd sums.

        clr     b        x:(r2)-n2,x0     y:(r4)-,y0
;       rep     #15
        mac     x0,y0,b x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,b x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,b x:(r2)-n2,x0     y:(r4)-,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)-,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)-n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        mac     x0,y0,b x:(r2)+n2,x0     y:(r4)+,y0
        macr    x0,y0,b (r2)-                             ;set y to start

        move    b,x0                                      ;save odd sum
        add     a,b      #16,n5                           ;even + odd
        sub     x0,a b,x:(r5)+                            ;even - odd
                                                          ; & save the sum data
        move    a,x:(r1)-                                 ;save the diff data
_poly30

        move    (r5)+n5                                   ;set for next pass

        rts

        page
        title   'Poly Analyze one super block'

; This routine poly-analyzes 36 blocks consisting of 32 samples each.

; on entry
;       r0 = starting address of a block of 1152 data points
;       m0 = set appropriately (may be a mod buffer if needed)
;       r5 = starting address of the output buffer for results
;       m5 = set appropriately (may be a mod buffer if needed)

; on exit
;       a = destroyed
;       b = destroyed
;       x0 = destroyed
;       y0 = destroyed
;       x1 = destroyed
;       y1 = destroyed
;       r0 = destroyed
```

```
;       r2 = destroyed
;       r3 = destroyed
;       r4 = destroyed
;       r5 = destroyed
;       n0 = destroyed
;       n1 = destroyed
;       n2 = destroyed
;       n3 = destroyed
;       n4 = destroyed
;       n5 = destroyed

        org     phe:

polyanal
        move    #63,m2                  ;set x buffer to mod 64
        move    #511,m3                 ;mod 512 buffer
        do      #36,_poly_44            ;do entire super-block
        jsr     panalysi                ;filter the data
        move    #63,m2                  ;set x buffer to mod 64
_poly_44

        move    #-1,m2                  ;restore m2
        move    #-1,m3                  ;restore m3

        rts

        page
; This function initializes the polyphase filter.
; It turns of the interrupt system for 512 cycles so beware.

; on entry
;       r0 = address of the X buffer for the analysis filter

; on exit
;       r0 = destroyed
;       a = destroyed

        org     phe:

polyaini
        clr     a
        rep     #512
        move    a,x:(r0)+

        rts
```


BAD ORIGINAL

```
        opt     fc
;
; c 1991. Copyright Corporate Computer Systems. Inc. All rights reserved.
;
; UXCODE\mem.asm

        title   'Relocatable Memory Declarations'

        include 'def.asm'

        section phase21
        xdef    Tonals,Maskers

        org     lhe:
stphase21_lhe

Tonals  ds      MAXTONALS*TONALSSIZE                        ;tonal array
Maskers ds      (MAXTONALS+NUMMAXCRITBNDS)*MASKERSSIZE      ;masker array-2 chans

endphase21_lhe
        endsec

        section phase2x
        xdef    GlbMsk
        xdef    Alising

        org     xhe:
stphase2x_xhe

GlbMsk          ds      MAXNMSKFREQS*2          ;global masking array
Alising         ds      MAXTONALS*ALIASSIZE*2   ;aliasing buffer

endphase2x_xhe
        endsec

        section NoisePwr
        xdef    NoisePwr

        org     lhe:
stNoisePwr_lhe

NoisePwr        ds      NUMMAXCRITBNDS          ;noise array

endNoisePwr_lhe
        endsec

        section b_i
        xdef    b_ii

        org     yhe:
stb_i_yhe

b_ii    ds      512

endb_i_yhe
        endsec

        section xtables
        xdef    Thres10SLB
        xdef    ThresSLB
```

```
        org     xhe:
stThr10SLB_xhe

;Threshold of hearing 10 dB down

Thres10SLB      ds      512

endThr10SLB_xhe
stThrSLB_xhe

;standard Threshold of hearing

ThresSLB        ds      512

endThrSLB_xhe
        endsec

        section ytables
        xdef    fmap_x
        xdef    cb_16k
        xdef    cb_22k
        xdef    cb_24k
        xdef    cb_32k
        xdef    cb_44k
        xdef    cb_48k
        xdef    g_cb_16k
        xdef    g_cb_22k
        xdef    g_cb_24k
        xdef    g_cb_32k
        xdef    g_cb_44k
        xdef    g_cb_48k
        xdef    bereich
        xdef    SubBandMap

        org     yhe:
stfmap_yhe

        include '..\uxcode\fmap.asm'     ;frequency mapping

endfmap_yhe

stcb_yhe

        include '..\uxcode\cb.asm'       ;noise tables

endg_cb_yhe

stbereich_yhe

        include '..\common\bereich.asm'

endbereich_yhe

stsbmap_yhe

        include '..\xlpsycho\sbmap.asm'  ;sub-band mapping

endsbmap_yhe
```

```
        endsec

        section codepass
        xdef    SBMsr
        xdef    SBMNRmax
        xdef    MNRval
        xdef    SBIndx
        xdef    SBPos
        xdef    AtLimit
        xdef    SvUsedSBs
        xdef    MNRsbc

        org     xhe:
stcodepass_xhe

;This array holds the MinMaskingDb - SubBandMax for each of the
;64 left (0-31) and right (32-63) subbands
;another 32 (64-95) are included as the joint channel array for allocation

SBMsr           ds      NUMSUBBANDS*3   ;Mask to Signal ratio by sub-band

;This array holds the deallocation selection values:
;       (MinMaskingDb - SubBandMax) - SNR[position at next lower index]
;for each of the 64 left (0-31) and right (32-63) subbands

SBMNRmax        ds      NUMSUBBANDS*2   ;Mask-to-Signal ratio + SNR[PrevPos]

;This array is for deallocation based on the least damage and has the
;sub-band values at the next lower position ordered over the 2 channel
;range of sub-bands. This array is paralled with the MNRsbc array below.

MNRval  ds      NUMSUBBANDS*2   ;table of ordered values (sub-band/chan)

; these arrays are dimension by *2 providing for the left channel
;  followed by the right channel

SBIndx          ds      NUMSUBBANDS*2   ;sub-band index
SBPos           ds      NUMSUBBANDS*2   ;sub-band positions left & right

;flags set when a sub-band reaches its limit of allocation:
;   (one per left channel for 32 subbands
;        and one per right channel for 32 sub-bands)
;       bit 0: set if below the global masking threshold
;       bit 1: set if not used or fully allocated

AtLimit ds      NUMSUBBANDS*2

;The SvUsedSBs array is for restoration prior to a required
;    joint stereo allocation.
;It is the saved array for the counters for sub-bands with assigned indices
;If a sub-band starts out below the Global Masking Threshold it takes
;a certain number of consecutive frames before it is skipped. Until that
;count down (SUBBANDSCTDOWN) reaches zero, the sub-band will receive at
;least one allocation.

SvUsedSBs       ds      NUMSUBBANDS*2

;This array is for deallocation based on the least damage and has the
;control info identifying sub-band number and channel of the ordered values
;in the MNRval array above.
```

```
MNRsbc  ds      NUMSUBBANDS*2    ;table of associated sub-bands/channel:
                                 ;       sub-bands  0-31 (bits 0 thru 4)
                                 ;       channel flagged by bit 6:
                                 ;               0 = left
                                 ;               1 = right

endcodepass_xhe
        endsec

        section arrays
        xdef    MinMskDb
        xdef    SBMaxDb
        xdef    SBits
        xdef    SBndSKF

        org     xhe:
starrays_xhe

; these arrays are dimension by *2 providing for the left channel
;   followed by the right channel

MinMskDb        ds      NUMSUBBANDS*2   ;minimum masking level in slb's
SBMaxDb         ds      NUMSUBBANDS*2   ;the maximum in each subband

; these arrays are dimension by *2 providing for the left channel
;   followed by the right channel

SBits           ds      NUMSUBBANDS*2   ;the S Bit array (scale factor type)
SBndSKF         ds      NUMSUBBANDS*NPERGROUP*2 ;sub-band scale factors

endarrays_xhe
        endsec

        section jntdata
        xdef    JntPlAnal
        xdef    JntSBits
        xdef    JntSBSKF
        xdef    JntSBMaxi

        org     xhe:
startjntdata_xhe

;these arrays are developed for handling joint stereo which is the
;combining of the left and right channel values

JntPlAnal       ds      INPCM           ;joint averaged left + right samples
JntSBits        ds      NUMSUBBANDS*2   ;the S Bit array (for joint stereo)
JntSBSKF        ds      NUMSUBBANDS*NPERGROUP*2 ;scale factors (joint stereo)
JntSBMaxi       ds      NUMSUBBANDS*NPERGROUP   ;joint Maxi scale factors

endjntdata_xhe
        endsec

        section highmisc
        xdef    IvSKF

        org     lhe:
stivskf
```


BAD ORIGINAL

```
        include '...uxcode-ivskf.asm'

endivskf
        endsec
```

```
        opt     cex
;
; (c) 1993. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; UXCODE\logpow.asm

        title   'convert to power and discrambling'

; This routine converts the amplitude data to power.
; The input to this routine is output of the real 1024 point FFT.  The
; FFT output is in scrambled order.  The output of the FFT should be such
; that the first output point is dc. The next output point corresponds to 46
; Hz, the next to 92 Hz ....

; This routine arranges the output of the fft in normal order and computes the
; power (amplitude squared).  The output of the fft corresponds to a real and
; an imaginary data point.  The power is computed as follows.
;
;       Pow(i) = Real(i) * Real(i) + Imag(i) * Imag(i)
;
; The real and imaginary parts of the data are stored in x an y memory
; respectively.
;
; Old versions of the psychoacoustic software ignored the dc value. Currently
; it is still that way. It should just 0 the dc value in the future.

; This routine reads its input from r0 and places the output at
; $800.  The output address is hardwired!!!!
; See Ben if you want to change the output address.

        org     pli:

logpow
; Also remenber that the memory is external so it is better to
; fetch from x and y memory in sperate instructions if possible

; on entry
;       r0 = address of the data to convert to power
;       index1 = address in y memory of descrambling table 1
;       index2 = address in y memory of descrambling table 2

; on exit
;       a = destroyed
;       b = destroyed
;       x0 = destroyed
;       y0 = destroyed
;       n4 = destroyed
;       n5 = destroyed
;       n6 = destroyed
;       r0 = destroyed
;       r1 = destroyed
;       r2 = destroyed
;       r3 = destroyed
;       r4 = destroyed
;       r5 = destroyed
;       r6 = destroyed

        move    #1,n6
        move    r0)-
        move    n6,n5
```


BAD ORIGINAL

```
        move    #3,n0
        move    #index1,r2                                ;discrambling table 1
        move    #index2,r3                                ;discrambling table 2
        move    n0,n1
        lua     r0)+,r1
        move    #$9ff,r6                                  ;calculate first two points

        move    x:(r1),y0
        move    y:(r0),x0
        mpy     x0,x0,a         x:(r0)+n0,x0
        mac     x0,x0,a
        move    a,l:(r6)

        mpy     y0,y0,b         y:(r1)+n1,y0
        mac     y0,y0,b         #$8ff,r6
        move    #1,n0
        move    b,l:(r6)

        move    r0,r4
        move    n0,n4

        do      #8,_discram1                                ;calculate the rest

        do      n0,_discram2

        move    x:(r0)+,x0       y:(r4)+,y0
        mpy     x0,x0,a          y:(r2)+,r6
        mac     y0,y0,a          x:(r0)+,x0       y:(r4)+,y0

        move    y:(r3)+,r5
        move    (r6)-n6

        mpy     x0,x0,b          (r5)-n5
        mac     y0,y0,b          a,l:(r6)
        move    b,l:(r5)
_discram2

        move    n0,b             ; multiply n0 by 2
        lsl     b
        move    b1,n0

        move    n0,n4
        move    (r0)+n0
        move    (r4)+n4
_discram1

        rts

        end
```

```
        opt     fc
;
;  (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UNCODE jntquant.asm

        title   'Joint Stereo Quantize Data'

; This routine is used to quantize the data using the Joint values
; MaxiFactor applicable to the current sub-band and block of 12 samples.
; The resulting data is right justified in the result register.
;
; This routine takes 64 - 2*number_of_bits cycles

; on entry
;       a = value
;       y:MaxiFact = the scale factor Maxi for sub-band and block
;       x0 = quantizing A value
;       x1 = quantizing B value
;       n4 = number of bits (1 - 16)
;!!!!!!!!!!!!! n4 must never be 0 or negative !!!!!!!!!!!!!!!!!!!!!!!!!

; on exit
;       a1 = result
;
;       a2 = destroyed
;       a0 = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r4 = destroyed

        include 'def.asm'

        section highmisc
        xdef    lshftbl

        org     yhe:
stjntquant_yhe

lshftbl
        dc      $000000                 ;bits =  0, place holder
        dc      $100000                 ;bits =  1, shift left 23 bits
        dc      $080000                 ;bits =  2, shift left 22 bits
        dc      $040000                 ;bits =  3, shift left 21 bits
        dc      $020000                 ;bits =  4, shift left 20 bits
        dc      $010000                 ;bits =  5, shift left 19 bits
        dc      $008000                 ;bits =  6, shift left 18 bits
        dc      $004000                 ;bits =  7, shift left 17 bits
        dc      $002000                 ;bits =  8, shift left 16 bits
        dc      $001000                 ;bits =  9, shift left 15 bits
        dc      $000800                 ;bits = 10, shift left 14 bits
        dc      $000400                 ;bits = 11, shift left 13 bits
        dc      $000200                 ;bits = 12, shift left 12 bits
        dc      $000100                 ;bits = 13, shift left 11 bits
        dc      $000080                 ;bits = 14, shift left 10 bits
        dc      $000040                 ;bits = 15, shift left 09 bits
        dc      $000020                 ;bits = 16, shift left 08 bits

endjntquant_yhe
        endsec
```


BAD ORIGINAL

```
;       section jntquant                          ;new
;       xdef    jntquantize                       ;new

;       org     p:$20                             ;new
;;      org     pli:                              ;rev 1
        org     phe:                              ;rev 1

jntquantize
        move    y:MaxiFact,y0                     ;get the Maxi scale factor
        tst     a         #lshftbl,r4             ;see if dividend is negative
        jlt     _quan_10                          ;it is

; - dividend and - divisor

        move    y:(r4+n4),y1
        and     #$fe,ccr                          ;clear the carry bit
        rep     n4                                ;value/scalefactor
        div     y0,a
        div     y0,a                              ;one more div
        div     y0,a                              ;one more div

        move    a0,y0                             ;get result to a reg
        mpy     y0,y1,a #qstbl,r4                 ;left justify
        jmp     _quan_20

; - divedend and - divisor

_quan_10
        neg     a       y:(r4+n4),y1              ;make +
        and     #$fe,ccr                          ;clear the carry bit
        rep     n4                                ;value/scalefactor
        div     y0,a
        div     y0,a                              ;one more div
        div     y0,a                              ;one more div

        move    a0,y0                             ;get result to a reg
        mpy     -y0,y1,a        #qstbl,r4         ;left justify

_quan_20
        move    a0,a
        tfr     x1,a    a,y0
        mac     x0,y0,a         y:(r4+n4),y1      ;form quantized result
        asr     a                                 ;divide by 2
        move    a,y0
        mpy     y1,y0,a                           ;right justify the bits

_quan_30
        rts
;       endsec                  ;new
```


BAD ORIGINAL

```
        opt fc
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; UNCODE irq.asm

        title 'IRQ interrupt handler for frame time interval timer'

; UNCODE irq.asm: XPSYCHO and XCODE combined dsp

        include 'def.asm'

;these save variables for exclusive use by interrupt irq handlers only

        section highmisc
        xdef    irqbR6Save
        xdef    irqbN6Save
        xdef    irqbM6Save
        xdef    irqbR7Save
        xdef    irqbN7Save
        xdef    irqbM7Save

        org     xhe:
stirq_xhe

irqbR6Save      ds      1
irqbN6Save      ds      1
irqbM6Save      ds      1
irqbR7Save      ds      1
irqbN7Save      ds      1
irqbM7Save      ds      1

endirq_xhe
        endsec

        org     phe:

; This little gem must be executed as a slow interrupt routine since
; it modifies the carry bit.

irqa
;       bset    #0,y:<qtalloc
        rti

; This routines swaps the processing addresses for the next frame to be
; processed: first by the XPSYCHO code and then formatted by the XCODE code
; This routine must have a higher priority than the ssi routines
; so r7 won't be changed by the ssi routine.
;
; This interrupt occurs each time an output buffer is done. This occurs
; each:         24 ms for 48k sampling
;               26.12244898 ms for 44.1K sampling
;               36 ms for 32k sampling
;               48 ms for 24K sampling
;               52.24489796 ms for 22.05K sampling
;               72 ms for 16K sampling

irqb
        move    r7,x:irqbR7Save         ;save the register
        move    m7,x:irqbM7Save         ;save m7
```

```
        move    n7,x:irqbN7Save             ;save n7
        move    r6,x:irqbR6Save             ;save the register

;since the T-MUSICAM frame pulse is aligned with the left channel pulse
;  align the A-to-D converter input to left channel, if needed

        move    y:<ipwptr,r7                ;next address for input pcm data
        move    #INPCM*2,n7                 ;parameter for 2 channel of values
        move    #PCMSIZE*2-1,m7             ;set as a mod buffer-2 channels

;if next address is to save a left channel value (on an even address), continue
;  else, adjust the odd address by 1 to make it an even address to start
;         the next frame properly capture values aligned left and right channel

        jclr    #0,y:<ipwptr,_irqb_00       ;if aligned for left channel, continue
        move    (r7)-                       ;back up to even address
        move    r7,y:<ipwptr                ;save aligned addr for next frame input

_irqb_00

;set input PCM sample address to start poly analysis of the frame just captured

        move    (r7)-n7                     ;back to start of just completed frame
        move    r7,x:<polyst                ;set current frame input buffer address

;shift for next frame to encode with the poly analyzed data

        move    y:<frmstrt,r7               ;get current frame start to output
        move    y:<frmnext,r6               ;addr of next frame buffer to encode
;;      move    y:<outsize,m7               ;circular buffer (2 frames worth)
        move    r6,y:<frmstrt               ;swap next buffer to current to encode
;!!!h221
        move    #reedsolomon,r6             ;to see if reed solomon frames
        move    y:<outsize,m7               ;circular buffer (2 or 3 frames worth)
        move    y:<outmus,n7                ;length of a frame
        jclr    #0,y:(r6),_not_reed         ;if not reed solomon, addr all set
        move    y:<frmnext,r7               ;addr of next frame buffer to encode
        nop
        move    (r7)+n7                     ;output frame ahead of one to be coded
_not_reed
;!!!h221
        move    r7,y:<oprptr                ;set where to start output read from
;;      move    r7,y:<frmnext               ;set next address for output
;;      move    (r7)-                       ;set last word of current frame
;;      move    r7,y:<frmlast               ;save last word addr for block seq numb

        bset    #0,y:<timer                 ;flick the timer sensed flag

        move    x:irqbR6Save,r6             ;restore the register
        move    x:irqbN7Save,n7             ;restore the n7
        move    x:irqbM7Save,m7             ;restore the m7
        move    x:irqbR7Save,r7             ;restore the register

        nop

        rti

;!!!debug unexpected interrupts

        org     phe:
```

```
stack_error
        nop
        nop
        nop
        rti

sciidle_line
        nop
        nop
        nop
        rti

scitimer
        nop
        nop
        nop
        rti

illegal_inst
        nop
        nop
        nop
        nop
        nop
        rti
```

```
        opt fc
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\hstvctor.asm

        title 'Host Vector receive data interrupts handler'

        include '..\common\ioequ.asm'

; Host Vector Receiver interrupts
;       read in the vector
;       if the receive data register not empty,
;               read in the vactor again

;save the host vectors for processing the switches next time

        section highmisc
        xdef    host24_word
        xdef    host26_word
        xdef    host28_word
        xdef    host2A_word
        xdef    host2C_word
        xdef    host2CA0Save
        xdef    host2CA1Save
        xdef    host2CA2Save
        xdef    host2CX0Save
        xdef    host2CR0Save
        xdef    host2CN0Save
        xdef    host2CM0Save
        xdef    host2E_word
        xdef    host30_word

        org     yhe:
sthstvctor_yhe

host24_word     ds      1
host26_word     ds      1
host28_word     ds      1
host2A_word     ds      1
host2C_word     ds      1
host2CA0Save    ds      1
host2CA1Save    ds      1
host2CA2Save    ds      1
host2CX0Save    ds      1
host2CR0Save    ds      1
host2CN0Save    ds      1
host2CM0Save    ds      1
host2E_word     ds      1
host30_word     ds      1

endhstvctor_yhe
        endsec

        org     phe:

;get the encoder switches host vector

hostvector_24
```

```
        movep   x:<<  _HRX,y:host24_word
        jset    #M_HRDF,x:<<M_HSR,hostvector_24

        rti

;get the encoder framing type host vector

hostvector_26

        movep   x:<<M_HRX,y:host26_word
        jset    #M_HRDF,x:<<M_HSR,hostvector_26

        rti

;get the encoder iso header host vector

hostvector_28

        movep   x:<<M_HRX,y:host28_word
        jset    #M_HRDF,x:<<M_HSR,hostvector_28

        rti

;get the psycho table offset ID for a new parameter value

hostvector_2A

        movep   x:<<M_HRX,y:host2A_word
        jset    #M_HRDF,x:<<M_HSR,hostvector_2A

        rti

;update the psycho table with a new parameter value

hostvector_2C

        movep   x:<<M_HRX,y:host2C_word
        jset    #M_HRDF,x:<<M_HSR,hostvector_2C

;save registers needed

        move    a0,y:host2CA0Save
        move    a1,y:host2CA1Save
        move    a2,y:host2CA2Save
        move    x0,y:host2CX0Save
        move    r0,y:host2CR0Save
        move    n0,y:host2CN0Save
        move    m0,y:host2CM0Save

;update the entry in the table

        move    #ptable,r0              ;address of the psycho parameter table
        move    #-1,m0                  ;set to a linear buffer

;see if table entry offset is valid (0 thru 31)

        move    y:host2A_word,a         ;get the table entry offset
        move    #0,x0                   ;to test for greater than 0
        cmp     x0,a    #>31,x0         ;see if less than 0
                                        ; & get set to test upper limit offset
```

```
        jlt     _host_2C_100                ;if less than 0, ignore the value
        cmp     x0,a     a,n0               ;see if offset to big
                                            ; & set addr offset into the table
        jgt     _host_2C_100                ;if to big an offset, ignore the value

;insert the new table value into active table and update sample rate table

        move    y:host2C_word,x0            ;get the parameter value
        move    x0,y:(r0+n0)                ;insert into its table entry
        move    y:psychaddr,r0              ;address of sample rate psycho table
        nop
        move    x0,y:(r0+n0)                ;insert into sample rate table entry

_host_2C_100

;restore the registers

        move    y:host2CA0Save,a0
        move    y:host2CA1Save,a1
        move    y:host2CA2Save,a2
        move    y:host2CX0Save,x0
        move    y:host2CR0Save,r0
        move    y:host2CN0Save,n0
        move    y:host2CM0Save,m0

        rti

;clean host vector buffer: read double buffer

hostvector_2E

        movep   x:<<M_HRX,y:host2E_word
        movep   x:<<M_HRX,y:host2E_word
        jset    #M_HRDF,x:<<M_HSR,hostvector_2E

        rti

;indicate to the host that the encoder interrupts are on and functioning

hostvector_30

        movep   x:<<M_HRX,y:host30_word
        jset    #M_HRDF,x:<<M_HSR,hostvector_30

        rti
```

```
                opt     cex
;
;  c  1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UKCODE\hanning.asm
                title   'hanning window'

; This function is used to apply a hanning window on the data.
; The window coeficients are symetric, ie 0 = 1023, 1 = 1022, ... .
; Thus the storage of the coefs is only 512 points.

; The window is applied to the x data (real) and the y data (imag)
; is set to all zero.

; The input data is stored oldest data at the lowest memory location.
; To read the data in temporal order, you must increment the location
; counter.

; on entry
;       r0 = address of the oldest input data in x memory
;       n0 = 2 to offset by channel count
;       r1 = address of the output data (real in x memory, imag in y)
;       m0 = set to right value to read input data

; on exit
;
;       r0 = destroyed
;       r1 = destroyed
;       r4 = destroyed
;       a = destroyed
;       x0 = destroyed
;       y0 = destroyed

        section hanning
        xdef    hcoef

        org     yhe:
sthcoef_yhe

        include '..\xlpsycho\hanwin.asm'

endhcoef_yhe
        endsec

        org     pli:
hanning
        move    #hcoef,r4                                   ;addr of window
        nop

        clr     a       x:(r0)+n0,x0    y:(r4)+,y0          ;window data
        do      #511,_han_10
        mpyr    x0,y0,a x:(r0)+n0,x0    y:(r4)+,y0          ;window data
        move    a,x:(r1)+
_han_10

        move    (r4)-
        mpyr    x0,y0,a x:(r0)-n0,x0    y:(r4)-,y0
        move    a,x:(r1)-
```

```
; do last 512

        do      #512,_han_20
        mpyr    x0,y0,a x:(r0)-n0,x0     y:(r4)-,y0         ;window data
        move    a,x:(r1)-
_han_20

; zero the y part

;       clr     a           (r1)-                           ;set to start addr
;       do      #1024,_han_30
;       move    a,y:(r1)-
;_han_30

        rts
```

```
        opt     fc,mex
;
;  c  1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  \XCODE\getsws.asm

        title   'Get encoder external switch settings'

; This routine is used to interpret the external switches on the box
;       defined for the encoder

; on exit
;       x:tstrate = raw bit rate input from the switches
;       x:tstfrme = framing mode input from the switches
;       x:tstband = sub-band width code input from the switches
;       x:tstbaud = ancillary data baud rate input from the switches
;       x:tstsel1 = application of line 1 select switch
;       x:tstsel2 = application of line 2 select switch
;       x:tstfrmt = frame communication formatting
;       x:tstreed = Reed/Solomon encoding switch
;!!!    x:tstbits = Reed/Solomon bit count to take from end of MUSICAM frames
;
;       y:<not_appl = bit 4 set if any switches changed
;
; destroyed:
;       register a

        include 'def.asm'
        include 'box_ctl.asm'

        section highmisc
        xdef    select1                 ;current setting of line 1 select switch
        xdef    select2                 ;current setting of line 2 select switch
        xdef    tstrate
        xdef    tstsmpl
        xdef    tstfrme
        xdef    tstband
        xdef    tstbaud
        xdef    tstoccs
        xdef    tstsel1
        xdef    tstsel2
        xdef    tstfrmt
        xdef    tstreed
;!!!    xdef    tstbits
        xdef    clntbits

        org     xhe:

stgetsws_xhe

select1         ds      1       ;current setting of line 1 select switch
select2         ds      1       ;current setting of line 2 select switch
tstrate         ds      1       ;raw bit rate input from the switches
tstsmpl         ds      1       ;raw sampling rate input from the switches
tstfrme         ds      1       ;raw framing mode input from the switches
tstband         ds      1       ;raw sub-band width code input from the switches
tstbaud         ds      1       ;raw ancil data baud rate input from switches
tstoccs         ds      1       ;MPEG-ISO (0) vs old CCS CDQ2000's (0)
tstsel1         ds      1       ;raw application of line 1 select switch
tstsel2         ds      1       ;raw application of line 1 select switch
```

```
tstfrmt         ds      1          ;raw frame communucation formatting
tstreed         ds      1          ;Reed/Solomon encoding switch
;!!!tstbits     ds      1          ;Reed/Solomon bit cnt from end of MUSICAM frames
clntbits        ds      1          ;client code defined as per client specs

endgetsws_xhe
        endsec

        org     phe:

getsws

        bclr    #4,y:<nct_appl  ;indicate no changes initially

        clr     a
        move    a,x:tstrate
        move    a,x:tstsmpl
        move    a,x:tstfrme
        move    a,x:tstband
        move    a,x:tstbaud
        move    a,x:tstoccs
        move    a,x:tstsel1
        move    a,x:tstsel2
        move    a,x:tstfrmt
        move    a,x:tstreed
;!!!    move    a,x:tstbits
        move    a,y:trailbits
        move    a,x:clntbits

;check the dip switches to determine frame bit rate
;  and ancillary data application and data baud rate

;switches that define the framing bit rate

        GET_BIT_RATE_CD

;switches that define the sampling bit rate

        GET_SAMPLE_RATE_CD

;switches that define the mode of framing: Stereo, Mono, Joint Stereo

        GET_FRAME_TYPE_CD

;switches that define the bit allocation sub-band width code

        GET_BAND_WIDTH_CD

;switches that define the  ancillary data baud rate

        GET_BAUD_RATE_CD

;switches to set if selecting line 1 and/or line 2

        GET_SELECTED_LINES_CD

;set client specified code for inclusion in frame

        GET_CLIENT_CODES_CD
```

```
;check for any changes in the control switches that would cause a restart

        move    x:tstrate,y1            ;look for a change in framing rate
        move    y:rawrate,a
        cmp     y1,a    x:tstsmpl,y1    ;set up to test sampling rate
        jne     _gsws_80
        move    y:smplrte,a
        cmp     y1,a    x:tstfrme,y1    ;set up to test framing mode
        jne     _gsws_80
        move    y:frmtype,a
        cmp     y1,a    x:tstband,y1    ;set up to test band width code
        jne     _gsws_80
        move    y:bndwdth,a
        cmp     y1,a    x:tstbaud,y1    ;set up to test ancillary data baud
        jne     _gsws_80
        move    y:baudrte,a
        cmp     y1,a    x:tstoccs,y1    ;set up to test MPEG-ISO vs old CCS
        jne     _gsws_80
        move    y:oldccs,a
        cmp     y1,a    x:tstsel1,y1    ;set up to test line 1 selection
        jne     _gsws_80
        move    x:select1,a
        cmp     y1,a    x:tstsel2,y1    ;set up to test line 2 selection
        jne     _gsws_80
        move    x:select2,a
        cmp     y1,a    x:tstfrmt,y1    ;set up to test framing format
        jne     _gsws_80
        move    y:frmformat,a
        cmp     y1,a    x:tstreed,y1    ;set up to test Reed/Solomon switch
        jne     _gsws_80
        move    y:reedsolomon,a
        cmp     y1,a
        jeq     _gsws_90

_gsws_80
        bset    #4,y:<not_appl          ;indicate changes in external switches

_gsws_90
        rts
```

```
        opt     fc
;
;  c. 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UKCODE\flushfrm.asm

; This routine outputs zero bits to the frame buffer
;       up to the bit count passed in.
; If the bit count has been passed already, an error condition is returned

; on entry
;       r0 = bit count to be zeroed up to
;       r4 = addr of reed solomon flag to skip flushing out the frame
;       y:<bitscnt = count of bits output to frame so far by setvalue rtn

; on exit
;       a = destroyed
;       b = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r0 = destroyed
;       r4 = destroyed
;       n4 = destroyed

        include 'box_ctl.asm'

        org     phe:

flushframe

;  unless these are reed solomon frames,
;   in that case, skip over the frame flush at this point

        clr     a                       ;set for OK return
        jset    #0,y:(r4),_flsh_90      ;if reed solomon, skip flush

_flsh_00

;pad 0 bits up to the bit count

        move    r0,a                    ;get bit count to end zero fill
        move    y:<bitscnt,y0           ;get count of bits put into frame so far
        cmp     y0,a                    ;see if end bit position reached
        jeq     _flsh_90                ;we've reached the end

;JUST IN CASE: see if the coded data is TOO LONG !!!!!!!!!

        jlt     _FLSH_HELP              ;IF WE OVERSHOT END OF FRAME ????????

;subtract the bits output so far

        sub     y0,a    #>16,y1         ;subtract bitscnt from bit total
                                        ; & set to zero 16 bits at a time

;see if a full 16 bits can be zeroed, else do remainder

        cmp     y1,a    a,n4            ;see if full 16 bits fit
                                        ; & set remainder bit len for setvalue
        jle     _flsh_10
```

```
;full 16 bits can be zeroed

        move    y1,n4                   ;set 16 bit length for setvalue

_flsh_11

;output zero bits to the end of the frame

        move    #0,y0
        jsr     setvalue

;go back and see if more bits to zero

        jmp     _flsh_00

_FLSH_HELP

;ERROR!!! this case should not occur

        ON_BITALLOC_LED_CD              ;!!! error we've overshot
        move    #>-1,a                  ;indicate the error

;!!!debug: dump the frame in question (pull of the ';' from next line)

;       jsr     dumpdata

_flsh_90

        rts
```

```
        opt     fc
;
;  (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UKCODE\findnois.asm

        title   'find noise maskers'
;
; This routine is used to find the noise maskers.  It does this by adding
; the power within a critical band.
;
; There will be MAXCRITBNDS values put into the NoiseDb array. The values
; are the power in watts.
;
; on entry
;       r1 = address of the power array (l memory)
;       r2 = address address of the noise array (l memory)
;
; on exit
;       a = destroyed
;       b = destroyed
;       b = destroyed
;       r1 = destroyed
;       r2 = destroyed
;       r3 = destroyed

        include 'def.asm'

        org     phe:

findnois

        move    y:cb,r3                 ;get the critical band boundries

        do      y:<maxcritbnds,_find_90
        move    y:(r3)+,b               ;get the # of bins for crit band
        clr     a
        do      b,_find_80
        move    l:(r1)+,b
        add     b,a                     ;add the power

_find_80
        move    a,l:(r2)+               ;save the noise

_find_90
        rts
```


BAD ORIGINAL

```
        opt     fc,cex
;
;  (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  \DXCODE\findskf.asm
;
; This routine is used to determine the scale factors for the blocks
; and subbands.  The input poly phase data is assumed to be dimensioned
;
;               PolyData[NUMBLOCKS][NUMPERSUBBAND][NUMSUBBANDS]
;
; The resulting scale factors are assumed to be dimensioned
;
;               SubBandSCFs[NUMSUBBANDS][NUMBLOCKS]
;
; on entry
;       r0 = polydata starting address
;       r1 = SubBandSKFs starting address
;
; on exit
;       a = destroyed
;       b = destroyed
;       x0 = destroyed
;       x1 = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r0 = destroyed
;       r1 = destroyed
;       r3 = destroyed
;       r4 = destroyed
;       r5 = destroyed
;       r6 = destroyed
;       n3 = destroyed
;       n4 = destroyed
;       n5 = destroyed
;       n6 = destroyed

        include 'def.asm'

        org     phe:
findskf
        move    #multbl,n4              ;get mult by 3 table address
        move    #lowtbl,n5              ;get lower table boundry address
        move    #upptbl,n6              ;get upper table boundry address
        move    #>5,b                   ;lower boundry

        move    #NUMSUBBANDS,n3         ;get skip factor

        do      y:<usedsb,_find_90
        move    r0,r3

        do      #NPERGROUP,_find_80

;       find the max in the sub-band
; !!! the following only works if NUMPERSUBBAND is even !!!
; !!! should fix so assembler kicks out if NUMPERSUBBAND is odd !!!

        move    x:(r3)+n3,a             ;Maxi = *p++ (++ is by NUMSUBBANDS)
        move    x:(r3)+n3,x0            ;temp = *p++ (++ is by NUMSUBBANDS)
        do      #(NUMPERSUBBAND-2)/2,_find_20
```

```
        cmpm    x0,a      x:(r3)-n3,x1      ;Maxi - temp
                                            ; & fetch next value to check
        tlt     x0,a                        ;set new maximum if necessary
        cmpm    x1,a      x:(r3)-n3,x0      ;Maxi - temp
                                            ; & fetch next value to check
        tlt     x1,a                        ;set new maximum if necessary
_find_20
        cmpm    x0,a                        ;Maxi - temp
                                            ; & set val for lower limit
        tlt     x0,a                        ;set new maximum if necessary
        cmpm    b,a       #62,r4
        jlt     _find_60

; ::: end even NUMPERSUBBAND specific code

        abs     a         #20,r4            ;form absolute value
        rep     #21                         ;move max of 22 bits
        norm    r4,a                        ;normalize

        move    r4,r6
        move    r4,r5
        move    y:(r6+n6),y0                ;get upper bounder

        cmp     y0,a      y:(r4+n4),r4      ;see if in upper 1/3
                                            ; & mult r4 by 3
        jge     _find_60                    ;it is

        move    y:(r5+n5),y1                ;get lower boundry
        cmp     y1,a      (r4)+             ;see if in middle 1/3
                                            ; & add 1 to r4
        jge     _find_60                    ;it is
        move    (r4)+                       ;must be in lower 1/3

_find_60
        move    r4,x:(r1)+                  ;*SubBandSKFs++

_find_80
        move    (r0)+                       ;SubBand++

_find_90
        rts
```

```
;
; c. 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; UXCODE\dbadd.asm
;
; This routine add two numbers which are logs
;
;       C = A + B
;
;       C = 10.0 * log10( 10.0**(A/10.0) + 10.0**(B/10.0) )
;
;       Assume that A is bigger than B.
;
;       C = A + 10.0 * log10( 1.0 + 10.0**((B-A)/10.0) )
;
;       The term
;
;               10.0 * log10( 1.0 + 10.0**((B-A)/10.0) )
;
;       can be approximated by a table where A-B is the index into the
;       table (after appropriate scaling).  This works since A-B is
;       always guarenteed to be positive.
;
;       The table is set to be in .5 dB increments with a maximum
;       dB difference of 31.5 dB.  If the difference is greater than
;       31.5 dB, then A is returned with nothing added.
;
;       Similarly if B is bigger.
;
; on entry
;       a = a
;       x0 = B
;
; on exit
;       b = C
;
;       b = destroyed
;       x0 = destroyed
;       r6 = destroyed

        section ytables
        xdef    DbAddTbl_3db
        xdef    DbAddTbl_6db

        org     yhe:

DbAddTbl_3db
        dc      0.0156250       ; 3.0103000, DbDif =     0.0000000
        dc      0.0143647       ; 2.7674916, DbDif =    -0.5000000
        dc      0.0131788       ; 2.5390189, DbDif =    -1.0000000
        dc      0.0120666       ; 2.3247408, DbDif =    -1.5000000
        dc      0.0110269       ; 2.1244260, DbDif =    -2.0000000
        dc      0.0100580       ; 1.9377592, DbDif =    -2.5000000
        dc      0.0091579       ; 1.7643486, DbDif =    -3.0000000
        dc      0.0083242       ; 1.6037356, DbDif =    -3.5000000
        dc      0.0075543       ; 1.4554046, DbDif =    -4.0000000
        dc      0.0068452       ; 1.3187948, DbDif =    -4.5000000
        dc      0.0061939       ; 1.1933105, DbDif =    -5.0000000
        dc      0.0055971       ; 1.0783324, DbDif =    -5.5000000
        dc      0.0050516       ; 0.9732279, DbDif =    -6.0000000
```

```
        dc      0.0045540       ; 0.8773604, DbDif =     -6.5000000
        dc      0.0041010       ; 0.7900975, DbDif =     -7.0000000
        dc      0.0036895       ; 0.7108185, DbDif =     -7.5000000
        dc      0.0033163       ; 0.6389203, DbDif =     -8.0000000
        dc      0.0029784       ; 0.5738222, DbDif =     -8.5000000
        dc      0.0026730       ; 0.5149694, DbDif =     -9.0000000
        dc      0.0023972       ; 0.4618361, DbDif =     -9.5000000
        dc      0.0021485       ; 0.4139269, DbDif =    -10.0000000
        dc      0.0019245       ; 0.3707776, DbDif =    -10.5000000
        dc      0.0017230       ; 0.3319562, DbDif =    -11.0000000
        dc      0.0015419       ; 0.2970616, DbDif =    -11.5000000
        dc      0.0013792       ; 0.2657238, DbDif =    -12.0000000
        dc      0.0012333       ; 0.2376020, DbDif =    -12.5000000
        dc      0.0011024       ; 0.2123840, DbDif =    -13.0000000
        dc      0.0009851       ; 0.1897844, DbDif =    -13.5000000
        dc      0.0008800       ; 0.1695429, DbDif =    -14.0000000
        dc      0.0007860       ; 0.1514228, DbDif =    -14.5000000
        dc      0.0007018       ; 0.1352092, DbDif =    -15.0000000
        dc      0.0006265       ; 0.1207077, DbDif =    -15.5000000
        dc      0.0005592       ; 0.1077423, DbDif =    -16.0000000
        dc      0.0004991       ; 0.0961541, DbDif =    -16.5000000
        dc      0.0004453       ; 0.0858000, DbDif =    -17.0000000
        dc      0.0003973       ; 0.0765510, DbDif =    -17.5000000
        dc      0.0003545       ; 0.0682913, DbDif =    -18.0000000
        dc      0.0003162       ; 0.0609165, DbDif =    -18.5000000
        dc      0.0002820       ; 0.0543331, DbDif =    -19.0000000
        dc      0.0002515       ; 0.0484573, DbDif =    -19.5000000
        dc      0.0002243       ; 0.0432137, DbDif =    -20.0000000
        dc      0.0002000       ; 0.0385351, DbDif =    -20.5000000
        dc      0.0001784       ; 0.0343609, DbDif =    -21.0000000
        dc      0.0001590       ; 0.0306374, DbDif =    -21.5000000
        dc      0.0001418       ; 0.0273160, DbDif =    -22.0000000
        dc      0.0001264       ; 0.0243538, DbDif =    -22.5000000
        dc      0.0001127       ; 0.0217119, DbDif =    -23.0000000
        dc      0.0001005       ; 0.0193560, DbDif =    -23.5000000
        dc      0.0000896       ; 0.0172553, DbDif =    -24.0000000
        dc      0.0000798       ; 0.0153821, DbDif =    -24.5000000
        dc      0.0000712       ; 0.0137119, DbDif =    -25.0000000
        dc      0.0000634       ; 0.0122229, DbDif =    -25.5000000
        dc      0.0000566       ; 0.0108953, DbDif =    -26.0000000
        dc      0.0000504       ; 0.0097118, DbDif =    -26.5000000
        dc      0.0000449       ; 0.0086567, DbDif =    -27.0000000
        dc      0.0000401       ; 0.0077161, DbDif =    -27.5000000
        dc      0.0000357       ; 0.0068777, DbDif =    -28.0000000
        dc      0.0000318       ; 0.0061302, DbDif =    -28.5000000
        dc      0.0000284       ; 0.0054640, DbDif =    -29.0000000
        dc      0.0000253       ; 0.0048701, DbDif =    -29.5000000
        dc      0.0000225       ; 0.0043408, DbDif =    -30.0000000
        dc      0.0000201       ; 0.0038689, DbDif =    -30.5000000
        dc      0.0000179       ; 0.0034484, DbDif =    -31.0000000
        dc      0.0000160       ; 0.0030735, DbDif =    -31.5000000
endDbAddTbl_3db

DbAddTbl_6db
        dc      0.0312500       ; 6.0205999, DbDif =      0.0000000
        dc      0.0299710       ; 5.7741972, DbDif =     -0.5000000
        dc      0.0287294       ; 5.5349831, DbDif =     -1.0000000
        dc      0.0275250       ; 5.3029399, DbDif =     -1.5000000
        dc      0.0263576       ; 5.0780378, DbDif =     -2.0000000
        dc      0.0252271       ; 4.8602359, DbDif =     -2.5000000
```


BAD ORIGINAL

```
dc      0.0241332       ; 4.6494816,  DbDif  =     -3.0000000
dc      0.0230755       ; 4.4457116,  DbDif  =     -3.5000000
dc      0.0220537       ; 4.2488521,  DbDif  =     -4.0000000
dc      0.0210674       ; 4.0588188,  DbDif  =     -4.5000000
dc      0.0201159       ; 3.8755184,  DbDif  =     -5.0000000
dc      0.0191989       ; 3.6988482,  DbDif  =     -5.5000000
dc      0.0183157       ; 3.5286972,  DbDif  =     -6.0000000
dc      0.0174658       ; 3.3649468,  DbDif  =     -6.5000000
dc      0.0166484       ; 3.2074711,  DbDif  =     -7.0000000
dc      0.0158629       ; 3.0561379,  DbDif  =     -7.5000000
dc      0.0151086       ; 2.9108093,  DbDif  =     -8.0000000
dc      0.0143847       ; 2.7713422,  DbDif  =     -8.5000000
dc      0.0136904       ; 2.6375896,  DbDif  =     -9.0000000
dc      0.0130251       ; 2.5094004,  DbDif  =     -9.5000000
dc      0.0123878       ; 2.3866210,  DbDif  =    -10.0000000
dc      0.0117778       ; 2.2690950,  DbDif  =    -10.5000000
dc      0.0111942       ; 2.1566648,  DbDif  =    -11.0000000
dc      0.0106363       ; 2.0491716,  DbDif  =    -11.5000000
dc      0.0101031       ; 1.9464559,  DbDif  =    -12.0000000
dc      0.0095939       ; 1.8483585,  DbDif  =    -12.5000000
dc      0.0091079       ; 1.7547208,  DbDif  =    -13.0000000
dc      0.0086442       ; 1.6653850,  DbDif  =    -13.5000000
dc      0.0082020       ; 1.5801950,  DbDif  =    -14.0000000
dc      0.0077806       ; 1.4989964,  DbDif  =    -14.5000000
dc      0.0073790       ; 1.4216371,  DbDif  =    -15.0000000
dc      0.0069966       ; 1.3479674,  DbDif  =    -15.5000000
dc      0.0066326       ; 1.2778407,  DbDif  =    -16.0000000
dc      0.0062863       ; 1.2111132,  DbDif  =    -16.5000000
dc      0.0059569       ; 1.1476444,  DbDif  =    -17.0000000
dc      0.0056436       ; 1.0872974,  DbDif  =    -17.5000000
dc      0.0053459       ; 1.0299388,  DbDif  =    -18.0000000
dc      0.0050630       ; 0.9754391,  DbDif  =    -18.5000000
dc      0.0047943       ; 0.9236722,  DbDif  =    -19.0000000
dc      0.0045392       ; 0.8745164,  DbDif  =    -19.5000000
dc      0.0042970       ; 0.8278537,  DbDif  =    -20.0000000
dc      0.0040671       ; 0.7835700,  DbDif  =    -20.5000000
dc      0.0038491       ; 0.7415553,  DbDif  =    -21.0000000
dc      0.0036422       ; 0.7017035,  DbDif  =    -21.5000000
dc      0.0034460       ; 0.6639124,  DbDif  =    -22.0000000
dc      0.0032601       ; 0.6280838,  DbDif  =    -22.5000000
dc      0.0030838       ; 0.5941233,  DbDif  =    -23.0000000
dc      0.0029168       ; 0.5619402,  DbDif  =    -23.5000000
dc      0.0027585       ; 0.5314475,  DbDif  =    -24.0000000
dc      0.0026086       ; 0.5025620,  DbDif  =    -24.5000000
dc      0.0024666       ; 0.4752040,  DbDif  =    -25.0000000
dc      0.0023321       ; 0.4492969,  DbDif  =    -25.5000000
dc      0.0022048       ; 0.4247680,  DbDif  =    -26.0000000
dc      0.0020842       ; 0.4015475,  DbDif  =    -26.5000000
dc      0.0019702       ; 0.3795688,  DbDif  =    -27.0000000
dc      0.0018622       ; 0.3587684,  DbDif  =    -27.5000000
dc      0.0017600       ; 0.3390858,  DbDif  =    -28.0000000
dc      0.0016634       ; 0.3204631,  DbDif  =    -28.5000000
dc      0.0015719       ; 0.3028455,  DbDif  =    -29.0000000
dc      0.0014854       ; 0.2861806,  DbDif  =    -29.5000000
dc      0.0014036       ; 0.2704184,  DbDif  =    -30.0000000
dc      0.0013262       ; 0.2555117,  DbDif  =    -30.5000000
dc      0.0012531       ; 0.2414154,  DbDif  =    -31.0000000
dc      0.0011839       ; 0.2280865,  DbDif  =    -31.5000000

endDbAddTbl_6db
```

```
        endsec
;turned into a macro for gcalcgic.asm
;;
;;      org     phe:
;;
;;DbAdd
;;      tfr     a,b     y:dbaddtbl,r6   ;assume A is biggest - so save A
;;                                      ; & get table base address
;;      sub     x0,a    #>$182,y0       ;form A-B
;;                                      ; & get scale factor
;;      jge     _db10
;;
;;      neg     a       x0,b            ;make difference a positive number
;;                                      ; & B was bigger - so save B
;;_db10
;;      move    a,x0
;;      mpyr    x0,y0,a #>$40,x0        ;form index into db table
;;                                      ; & get upper limit + 1
;;      cmp     x0,a    a1,n6           ;see if within table range
;;                                      ; & set index in proper register
;;      jge     _db20
;;
;;      rnd     b       y:(r6+n6),x0    ;rnd of b not really necessary
;;                                      ; & get table entry
;;      add     x0,b
;;
;;_db20
;;      rts
```

```
;
;  © 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UXCODE\dbadd.mac

; DbAdd routine turned into a macro for qcalcglo.asm

        nolist

DBADD macro

        tfr     a,b       y:dbaddtbl,r6    ;assume A is biggest - so save A
                                            ; & get table base address
        sub     x0,a      #>$182,y0         ;form A-B
                                            ; & get scale factor
        jge     _db10

        neg     a         x0,b              ;make difference a positive number
                                            ; & B was bigger - so save B
_db10
        move    a,x0
        mpyr    x0,y0,a   #>$40,x0          ;form index into db table
                                            ; & get upper limit + 1
        cmp     x0,a      a1,n6             ;see if within table range
                                            ; & set index in proper register
        jge     _db20

        rnd     b         y:(r6+n6),x0      ;rnd of b not really necessary
                                            ; & get table entry
        add     x0,b

_db20
        endm
        list
```


BAD ORIGINAL

```
;
; c' 1994. Copyright Corporate Computer Systems. Inc. All rights reserved.
;
; UXCODE\fft.asm
;
; This program originally available on the Motorola DSP bulletin board.
; It is provided under a DISCLAMER OF WARRANTY available from
; Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
; Radix 2, In-Place, Decimation-In-Time FFT (smallest code size).
; (test program)
;
; Last Update 30 Sep 86   Version 1.1
;
        opt     nomd,mex,cre,nocex

        include '..\uxcode\fftr16b.asm'

        define points   '1024'
        define data     '$c00'                  ;hanning and fft buff
        define coef     '$400'                  ;for sine and cosine table
        define dacol    '$A00'                  ;for fft table

        org     pli:
fft
        fftr16b  points,data,coef,coef1,dacol

; clean up after the fft has done its work.
; This makes the fft routine a "nice" routine.

        move    #-1,m0
        move    m0,m1
        move    m1,m2
        move    m2,m3
        move    m3,m4
        move    m4,m5
        move    m5,m6

        rts
```

```
        opt     fc
;
;  c. 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UXCODE\crc.asm

        title   'CRC polynominal calculation'

; This routine computes the CRC of a string of bits in Y-Memory.
; It can use any arbitrary polynominal generator.
; It does this by simulating the hardware shift register implementation.
; This means that it does the calculation a bit at a time.
;
; X1 is set to indicate the number of bits the msb of the first data
; word is shifted. For example, setting x1 = 0 implies the the the first
; data value is left justified in the word.

; on entry
;       r0 = Y-memory address of data array, msb of data is in msb of the words
;       r1 = number of bits to checksum
;       x0 = check sum seed value: 'ffff00' or '000000'
;       x1 = bit offset of the first data bit from the msb position (0-23)
;       y1 = CRC generator polynominal left justified ($800500 for CRC-16)

; on exit
;       a1 = CRC right justified 16-bit value
;
;       a0 = destroyed
;       a2 = destroyed
;       b = destroyed
;       x0 = destroyed
;       x1 = destroyed
;       y0 = destroyed
;       y1 = destroyed

; register usage
;       y1 = CRC polynominal value
;       r0 = address of the next data word
;       r1 = number of bit left to work on
;       a1,a0 = current data value and data value + 1
;       x0 = accumulator (16 bit shift register - left justified)
;       b = general temp

        include '..\common\def.asm'

        section lowmisc
        xdef    crcstrt

        org     yli:
stcrc_yli

crcstrt ds      1                       ;flag to control step over checksum
                                        ; bit 0 - set after up to checksum
                                        ; bit 1 - set after stepping over check

endcrc_yli
        endsec

        org     phe:
```

```
crc
        move    y:(r0)-,a1                      ;get the first data word

        move    x1,b                            ;get any offset bits
        tst     b       y:(r0)-,a0              ;see if need to adjust the input data
                                                ; & get the second data word
        jeq     _crc_10                         ;no adjustment necessary

        rep     x1                              ;move the msb of data to top of a1
        asl     a                               ;shift into place

_crc_10
        move    #>24,b                          ;compute the number of bits for first lp
        sub     x1,b    b0,y:<crcstrt           ;number of bits remaining in 1st word
                                                ; & zero checksum itself avoidance ctl
        move    b,x1                            ;sum remaining bits in 1st word
        move    #CRC_STORED_BIT_OFFSET,r2       ;offset to the checksum stored
        move    #$800000,y0                     ;msb of  the accumulator

_crc_20
        do      x1,_crc_60
        jset    #1,y:<crcstrt,_crc_28           ;passed over checksun finished
        jset    #0,y:<crcstrt,_crc_26           ;stepping over checksun continues
        move    (r2)-                           ;count the bits processed
        move    r2,b                            ;test r2 reached 0 yet
        tst     b                               ;is last bit before checksum reached
        jne     _crc_28                         ;no, continue summing early bits
        bset    #0,y:<crcstrt                   ;flag to skip over checksum
        move    #NCRCBITS,r2                    ;bits in checksum to skip over
        jmp     _crc_28                         ;sum this last bit

_crc_26
        move    (r2)-                           ;count the bits processed
        move    r2,b                            ;test r2 reached 0 yet
        tst     b       (r1)-                   ;is last bit of checksum reached
                                                ; & decrement the bit ctr (r1)
        jne     _crc_45                         ;no, continue skipping over bits
        bset    #1,y:<crcstrt                   ;flag checksum skipped over
        jmp     _crc_45                         ;skip over this bit

_crc_28
        tfr     a,b                             ;save the current data in b
        eor     x0,b                            ;look at the lsb of both data and accum
        and     y0,b                            ;only the msb
        jpl     _crc_30                         ;if +, then no subtraction necessary

        move    x0,b                            ;get the accumulator
        asl     b       (r1)-                   ;and shift left 1 bit
                                                ; & decrement the bit ctr (r1)
        eor     y1,b                            ;and subtract the polynominal generator
        jmp     _crc_40

_crc_30
        move    x0,b                            ;get the accumulator
        asl     b       (r1)-                   ;shift the accumulator one bit left
                                                ; & decrement the bit ctr (r1)

_crc_40
        move    b1,x0                           ;save as the new accumulator
```

```
_crc_45
        asl     a       r1,b            ;shift the data one bit left
                                        ; & setup test # bits left to process
        tst     b                       ;see if any left
        jgt     _crc_50

        enddo                           ;all done

        move    #$008000,y0             ;shift value
        mpy     x0,y0,a #>$ffff,x0      ;right adjust the crc to lsb of a1
        and     x0,a                    ;remove the debris
        move    #0,a0                   ;remove the debris
        move    #0,a2                   ;remove the debris

        rts

_crc_50
        nop

_crc_60
        move    #>24,x1                 ;set next loop count
        move    y:(r0)-,a0              ;get the next word
        jmp     _crc_20

        end
```

```
        opt     fc
;
; c  1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; .UXCODE zeropowe.asm

        title   'zero power'

; This function is used to zero the power around a tonal.

; on entry
;       r1 = address of the power array (1 memory)
;       r2 = address of the tonal structure  1 memory)
;       r3 = number of tonals in the tonal structure
;       r4 = address of the range table  y memory)

; on exit
;
;       a = destroyed
;       b = destroyed
;       x0 = destroyed
;       y0 = destroyed
;       r1 = destroyed
;       r2 = destroyed
;       r5 = destroyed
;       n1 = destroyed
;       n2 = destroyed
;       n4 = destroyed

        include 'def.asm'

        org     phe:

zeropowe
        move    r3,a                            ;get number of tonals
        tst     a                               ;check if a good number
        jle     _zero_90

;       move    #0,a                            ;value to set power to

        move    #2,y0                           ;shift right 6 bits

        move    #TONALSBIN,n2                   ;get offset to the bin
        move    r1,r5                           ;save starting position
        move    (r2)+n2                         ;position to first bin

        move    #TONALSSIZE,n2                  ;now get the size of structure
        nop

        do      r3,_zero_90
        move    x:(r2)+n2,n1                    ;get the next bin number

        move    n1,a                            ;test for max bin number
        move    #>490,x0                        ;max bin number
        cmp     x0,a     #0,a                   ;see if at max
                                                ; & clear a to zero values
        jlt     _zero_30

;bin at max, break out of loop and exit
```


BAD ORIGINAL

```
        enddo
        jmp     _zero_90

_zero_00

;process bins not at max

        move    n1,x0

        mpy     x0,y0,b (r1)+n1                 ;shift right 6 bits
        move    b1,n4                           ;save the offeset into rngtbl
        nop
        move    y:(r4+n4),n1                    ;get the range
        move    n1,b                            ;save for later
        asl     b        #>1,x0
        add     x0,b     (r1)-n1                ;2 * range + 1

; use a do loop to keep interrupts alive.
; remember, a rep keeps interrupts off

        do      b,_zero_10
        move    a,l:(r1)-                       ;zero the power

_zero_10
        move    r5,r1                           ;restore starting array addr

_zero_90

        rts
```

```
        opt     fc
;
;  c  1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  \MXCODE\setckskf.asm

        title   'Set scale factor CRC checksums'

; This routine has the 4 scale factor check sums that apply to the
; current frame. They are then stored in the end of the previous frame that
; was just coded. These values are the last 48 bits (12 each) in the
; MUSICAM frame. They may be followed by any client reserved bits and
; the CCS CDQ2000 block serial number when in combined mode.
;       The check sums protect groups of scale factors by sub-band range:
;               1. sub-bands 0 thru 3
;               2. sub-bands 4 thru 7
;               3. sub-bands 8 thru 11
;               4. sub-bands 12 thru 31

; on exit
;       r0 = destroyed
;       r1 = destroyed
;       n1 = destroyed
;       r2 = destroyed
;       n2 = destroyed
;       r3 = destroyed
;       r4 = destroyed
;       a2 = destroyed
;       a1 = destroyed
;       b = destroyed
;       x0 = destroyed
;       y0 = destroyed
;       y1 = destroyed

        include 'def.asm'

        section highmisc
        xdef    private
        xdef    skfcrcwd
        xdef    skfcrcbt
        xdef    calskfck
        xdef    sbctls
        xdef    skfcnt1
        xdef    skfcnt2
        xdef    skfcnt3
        xdef    skfcnt4

        org     xhe:
stsetckskf_xhe

private dc      1               ;header indication of application
                                ;  0 not appl, 1 frame has checksums

skfcrcwd        ds      1       ;word at frame end for next frame skf checksums
skfcrcbt        ds      1       ;bit offset for next frame skf checksums

;calculated skf check sums

calskfck        ds      1       ;checksum 1  -1 initialize as none yet.
                ds      1       ;checksum 1  -1 initialize as none yet
```

```
                ds      1       ;checksum 2  -1 initialize as none yet!
                ds      1       ;checksum 3  -1 initialize as none yet!

sbctls                          ;table to control checksum:
                                ;   word one = starting sub-band
                                ;   word two = count of sub-bands
                dc      0       ;sub-band 0
skfcnt1         dc      4       ;     thru sub-band 3
                dc      4       ;sub-band 4
skfcnt2         dc      4       ;     thru sub-band 7
                dc      8       ;sub-band 8
skfcnt3         dc      4       ;     thru sub-band 11
                dc      12      ;sub-band 12
skfcnt4         dc      20      ;     thru sub-band 31  adjustable.

endsetckskf_xhe
        endsec

        org     phe:

;this routine calculates and stores the frame checksums
;if the private bit in the frame header is set to 1

setckskf

;if flag from frame header says skf checksums included (private bit = 1)
; retreive the scale factor checksum from the frame and save it

        move    #private,r2             ;to test for appliaction
        nop
        jclr    #0,x:(r2),_sskfcrc_900  ;if not appl, return

;if this is the 1st frame after a restart, can't store scale factor crc's

        jclr    #2,x:(r2),_sskfcrc_900  ;it's 1st frame, return

;replaced uncoded scale factors with '63'

        move    #SBIndx,r0              ;array of assigned indexes
        move    #SBndSKF,r1             ;array of scale factors
        move    #NPERGROUP,n1           ;3 scale factors per sub-band

;go through all indexes for left channel and set any unused sub-bands

        do      #NUMSUBBANDS,_sskfcrc_2
        move    x:(r0)+,a               ;get left channel sub-band index
        tst     a       #>63,x0         ;test for zero, set rpelace value
        jne     _sskfcrc_0              ;if used, adjust SBndSKF address

;replace 3 scale factors with 63

        move    x0,x:(r1)+              ;scale factor 1
        move    x0,x:(r1)+              ;scale factor 2
        move    x0,x:(r1)+              ;scale factor 3
        jmp     _sskfcrc_1

_sskfcrc_0

;scale factors should remain as is, bump up the address
```


BAD ORIGINAL

```
        move    (r1)-n1

_sskfcrc_1
        nop

_sskfcrc_2

;go through all indexes for right channel and set any unused sub-bands
;unless we have a JOINT stereo frame. In that case, do through sibound

        move    y:opfrtyp,b              ;frame type to see if joint
        move    #>JOINT_STEREO,x1        ;joint stereo code
        move    #SBIndx,r2               ;array of left channel assigned indexes
        cmp     x1,b    #NUMSUBBANDS,n2  ;see if joint
                                         ; & if case not, all right chan indexes
        jne     _sskfcrc_3               ;not joint, do all right channel indixes

;we have a joint frame, use only right channel real indexes
; set up the offset to the left channel indexes

        move    y:<sibound,y0            ;get boundary sub-band count
        move    #>NUMSUBBANDS,a          ;total sub-bands per channel
        sub     y0,a    y:<sibound,n2    ;calc remaining sub-bands
                                         ; & set 1st loop sub-bands to test
        move    a,y0                     ;save the remainig sub-band count
        move    (r2)+n2                  ;set addr to start left channel indexes

_sskfcrc_3

;set the required right channel indexes at zero

        do      n2,_sskfcrc_6
        move    x:(r0)+,a                ;get right channel sub-band index
        tst     a       #>63,x0          ;test for zero, set rpelace value
        jne     _sskfcrc_4               ;if used, adjust SBndSKF address

;replace 3 scale factors with 63

        move    x0,x:(r1)+               ;scale factor 1
        move    x0,x:(r1)+               ;scale factor 2
        move    x0,x:(r1)+               ;scale factor 3
        jmp     _sskfcrc_5

_sskfcrc_4

;scale factors should remain as is, bump up the address

        move    (r1)+n1

_sskfcrc_5
        nop

_sskfcrc_6

;if doing a joint frame right channels, do the rest based on the left channel
; indexes

        cmp     x1,b    r2,r0            ;see if joint
                                         ; & set the left channel start address
        jne     _sskfcrc_9               ;not joint, done
```

```
;we have a joint frame, use only right channel real indexes
; set up the offset to the left channel indexes
;set the required right sub-bands based on left channel indexes at zero

        do      y1,_sskfcrc_9
        move    x:(r0)-,a                 ;get left channel sub-band index
        tst     a        #>63,x0          ;test for zero, set rpelace value
        jne     _sskfcrc_7                ;if used, adjust SBndSKF address

;replace 3 scale factors with 63

        move    x0,x:(r1)-                ;scale factor 1
        move    x0,x:(r1)-                ;scale factor 2
        move    x0,x:(r1)-                ;scale factor 3
        jmp     _sskfcrc_8

_sskfcrc_7

;scale factors should remain as is, bump up the address

        move    (r1)-n1

_sskfcrc_8
        nop

_sskfcrc_9

;initialize the sub-band counts for the 4 CRC-checksums

        move    #>4,x0                    ;set sub-band cnt for 1st 3 groups
        move    x0,x:skfcnt1              ;set sub-band count group 1
        move    x0,x:skfcnt2              ;set sub-band count group 2
        move    x0,x:skfcnt3              ;set sub-band count group 3

;now make any adjustments if applicable MAXSUBBANDS is 12 or less

        move    y:<maxsubs,a              ;get applicable MAXSUBBANDS or testing
        move    #>12,x0                   ;total sub-band cnt for 1st 3 groups
        cmp     x0,a     #>0,x1           ;see if more than 12 MAXSUBBANDS
                                          ; & set to zero subs count if needed
        jgt     _sskfcrc_10               ;if so, go to set sub-band cnt group 4

;we have an applicable MAXSUBBANDS of 12 or 8
;  group 4 is not applicable

        move    x1,x:skfcnt4              ;zero the sub-band count for 4th group
        move    #>8,x0                    ;see if 3rd group gets zeroed also
        cmp     x0,a                      ;is applicable MAXSUBBANDS is 8
        jgt     _sskfcrc_11               ;if MAXSUBBANDS = 12, continue

;we have an applicable MAXSUBBANDS of 8
;  group 3 is not applicable

        move    x1,x:skfcnt3              ;set count of sub-bands in 3rd crc
        jmp     _sskfcrc_11

_sskfcrc_10

;set the 4th sub-band count based on applicable MAXSUBBANDS
```

```
;       move    y:<maxsubs,a                ;current MAXSUBBANDS
;       move    #>12,x0                     ;subtract 12 subs (for 1st 3 crcs)
        sub     x0,a                        ;set the sub-band count for 4th crc
        move    a,x:skfcnt4                 ;set count of sub-bands in 4th crc

_sskfcrc_11

;do the scale factor checksum checks

        move    #SBndSKF,r0                 ;addr of scale factors array
        move    #calskfck,r3                ;addr of calculated checksums
        move    #sbctls,r4                  ;addr if table to control checksum

;indicate whether 2 channels (n2 = 0) of mono (n2 = 1)

        move    #0,n2
        jclr    #STEREO_vs_MONO,y:<stereo,_sskfcrc_30
        move    #1,n2

_sskfcrc_30

;calculate and test scale factor checksums

        do      #NUMSKFCKSUMS,_sskfcrc_40
        move    x:(r4)+,b                   ;indicate starting sub-band number
        move    x:(r4)+,a                   ;set number of sub-bands included

;see if the sub-band count is zero, and if so, skip the scale factor checksum

        tst     a                           ;check sub-band count for zero
        jne     _sskfcrc_33                 ;if not zero, calc CRC checksum

;this sub-band group has no sub-bands, zero the CRC checksum

        clr     b                           ;set checksum to zero
        jmp     _sskfcrc_36                 ;store zero checksum

_sskfcrc_33

;calculate the checksum (result is returned in b1)

        jsr     crcskf

_sskfcrc_36

;store the scale factor checksum

        move    b1,b                        ;clean up checksum
        move    b,x:(r3)+                   ;save scale factor checksum

_sskfcrc_40

;set up the checksums for storing in the previous frame

        move    #private,r2                 ;to test type of prev frame
        move    #calskfck,r0                ;addr of calculated checksums
        move    #calskfck,r1                ;to store after alignment

;see if the previous frame was a split mono frame for bit duplication
```

```
        jset    #1,x:r2.,_sskfcrc_80

        move    #>2,y1                          ;store 2 formatted words

;not a split frame, concatenate pairs of 12-bit checksums into one 24-bit word

        do      y1,_sskfcrc_70

;get the next pair of checksums

        move    x:(r0)-,a
        move    x:(r0)-,b

;left justify the 2nd checksum of the pair

        do      #12,_sskfcrc_50
        asl     b

_sskfcrc_50

;concatenate right justified 1st checksum with left justified 2nd checksum

        move    b1,a0

;shift left pair of checksums into a1

        do      #12,_sskfcrc_60
        asl     a

_sskfcrc_60

        move    a1,x:(r1)+                      ;save the aligned pair

_sskfcrc_70

        jmp     _sskfcrc_140

_sskfcrc_80

;is a split frame, duplicate the 12-bit checksum bits into one 24-bit word

        move    #>4,y1                          ;store 4 formatted words

        do      y1,_sskfcrc_140

;clear the target register and get the checksum

        clr     b       x:(r0)-,a

;rotate right the bits from the right justified checksum and rotate them
;right in pairs into the target register

        do      #12,_sskfcrc_135
        ror     a                               ;bit to carry bit

;test the carry bit to see if zero or 1

        jcs     _sskfcrc_90                     ;if carry bit set, so indicate
        bclr    #10,y:<not_appl                 ;carry bit = 0
```

```
        jmp     _sskfcrc_100                    ;go duplicate the bit

_sskfcrc_90
        bset    #10,y:<not_appl                 ;carry bit = 1

_sskfcrc_100

;now output 2 copies of the bit in the target register

        do      #2,_sskfcrc_130
        jset    #10,y:<not_appl,_sskfcrc_110    ;test if bit zero or 1
        andi    #$FE,ccr                        ;bit is a zero
        jmp     _sskfcrc_120                    ;go insert the bit

_sskfcrc_110

        ori     #$01,ccr                        ;bit is a 1

_sskfcrc_120

;push the bit into the target register

        ror     b

_sskfcrc_130
        nop

_sskfcrc_135

;store the duplicated checksum for frame insertion

        move    b1,x:(r1)+

_sskfcrc_140

;now insert either the 2 or the 4 formatted words in the end of previous frame
;position to the scale factor checksums in the frame buffer

        move    x:skfcrcwd,r0           ;end of frame word address
        move    x:skfcrcbt,a            ;bit offset to start skf crc's
        move    y:<outsize,m0           ;circ buffer ctl
        move    #calskfck,r1            ;addr of formatted checksums
        move    y:(r0),b                ;word from prev frame to start insert
        tst     a       #>24,x0         ;see if a right justify shift is needed
                                        ; & set bits per word
        jeq     _sskfcrc_150            ;no need to shift the 1st word

        sub     x0,a                    ;get bits to shift formatted word
        neg     a                       ;bits to shift the word to receive

;shift the formatted word to be ready to receive skf crc's

        do      a,_sskfcrc_150
        asr     b       x:skfcrcbt,a    ;right justify bits
                                        ; & restore bit offset to start crc's

_sskfcrc_150

;for the number of formatted words, shift the bits into the previous frame
```

```
        do      y1,_sskfcrc_180
        move    x:(r1)+,b0                      ;get formatted checksum word

;insert the 24 bits

        do      #24,_sskfcrc_170

;see if full word shifted

        cmp     x0,a     #>1,y0                 ;test if 24 bits shiffted
                                                ; & set bit shift incrementer
        jlt     _sskfcrc_160                    ;if less than 24, shift bit in

;24 bits have been inserted, clear bits per word ctr an put word into prev frame

        clr     a                               ;zero bit shift counter
        move    b1,y:(r0)+                      ;formatted word to prev frame

_sskfcrc_160

;shift the bit into low order of word and count the bit inserted

        asl     b                               ;insert bit into b1
        add     y0,a                            ;increment word bit ctr

_sskfcrc_170
        nop

_sskfcrc_180

;see if bits from frame buffer need to be formatted
;if an exact word fit, insert the newly formatted word

        cmp     x0,a     a,y1                   ;see if 24 bits in word
                                                ; & save bits inserted count
        jeq     _sskfcrc_200                    ;if 24 bits, store the word

;we have to get the next word from the frame buffer and concatenate with the
;remaining bits of the scale factor checksums

        move    #>24,a                          ;to calc num bits to shift
        sub     y1,a                            ;determine bits to shift

;get the word at the next address

        move    a,y0                            ;save bits to left shift
        move    y:(r0),a                        ;get word from frame

;left shift the word from the frame same bits as shifted into b1

        do      y1,_sskfcrc_190
        asl     a                               ;left justify in a1

_sskfcrc_190

;now concatenate the right justified checksum in b1 wit left justified
; word from the frame

        move    a1,b0                           ;left justified frame word
```

```
;shift the remaining bits together for a fully formatted 24-bit word

        jc      y0,_sskfcrc_300
        asl     b

_sskfcrc_300

;store the 4th of the scale factor checksums in previous frame

        move    b1,y:(r0)                       ;word to frame buffer
        move    #-1,m0                          ;restore linear buffer ctl

_sskfcrc_900

;indicate that a frame has been formatted

        bset    #2,x:private

        rts
```

```
        opt     fc
;
;  c. 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  \XCODE\setbal.asm

        title   'Set the bit allocations'

; New ISO frame format for stereo (2/7/91)

; This routine outputs the bit allocation bits.
;
; It is the ISO standard.
; sub-band  0 - 10 use 4 bits (11 * 4 = 44 bits)
; sub-band 11 - 22 use 3 bits (12 * 3 = 36 bits)
; sub-band 23 - 26 use 2 bits ( 4 * 2 =  8 bits)
;                                total = 88 bits)
;
; on entry
;       r6 = current offset in output array
;       y:<maxsubs = encoded sub-band range at:
;               sampling rate, bit rate  and whether MONO or 2 channels
;       y:<sc = shift count
;       y:opfrtyp = full stereo, joint stereo or mono
;       y:<stereo = type of framing flags used:
;                       bit 0 means stereo vs mono framing
;                               0 = stereo framing
;                               1 = mono framing
;                       bit 2 is to simply indicate that joint stereo applies
;                               0 = NOT joint stereo framing type
;                               1 = IS joint stereo framing type
;                       bit 3 is to indicate the full stereo upgrade by allocate rtn
;                           if joint stereo applies
;                               0 = normal joint stereo allocation
;                               1 = FULL STEREO allocation
;                       bit 4 is to simply indicate the stereo intensity sub-band
;                            boundary has been reached if joint stereo applies
;                               0 = NO sub-bands still below intensity boundary
;                               1 = sub-bands above intensity boundary
;
;       y:<sibound = for joint stereo sub-band intensity boundary
;       x:crcbits = accumulator of bits covered by CRC-16 routine
;                       (bit allocation bits are accumulated)
;
;       r0 = address of left and right channels SubBandIndex array (x memory)

; on exit
;       a = destroyed
;       b = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r0 = destroyed
;       r1 = destroyed
;       r2 = destroyed
;       r4 = destroyed
;       n4 = destroyed

        include 'def.asm'

        section highmisc
```

```
        xdef    skftbl
        xdef    skftbl_1
        xdef    skftbl_2
        xdef    skftbl_3

        org     yhe:
stsetbal_yhe

;address of BAL's bit table as per Allowed table selected

skftbl  ds      1

;These tables is the number of bits used by the scale factor in each sub-band

; High sampling rates with higher bit rate framing

skftbl_1
        dc      4               ;sub-band 0
        dc      4               ;sub-band 1
        dc      4               ;sub-band 2
        dc      4               ;sub-band 3
        dc      4               ;sub-band 4
        dc      4               ;sub-band 5
        dc      4               ;sub-band 6
        dc      4               ;sub-band 7
        dc      4               ;sub-band 8
        dc      4               ;sub-band 9
        dc      4               ;sub-band 10

        dc      3               ;sub-band 11
        dc      3               ;sub-band 12
        dc      3               ;sub-band 13
        dc      3               ;sub-band 14
        dc      3               ;sub-band 15
        dc      3               ;sub-band 16
        dc      3               ;sub-band 17
        dc      3               ;sub-band 18
        dc      3               ;sub-band 19
        dc      3               ;sub-band 20
        dc      3               ;sub-band 21
        dc      3               ;sub-band 22

        dc      2               ;sub-band 23
        dc      2               ;sub-band 24
        dc      2               ;sub-band 25
        dc      2               ;sub-band 26
;end table 3-B.2a
        dc      2               ;sub-band 27
        dc      2               ;sub-band 28
        dc      2               ;sub-band 29
;end table 3-B.2b
        dc      2               ;sub-band 30
        dc      2               ;sub-band 31

; High sampling rates with lower bit rate framing

skftbl_2
        dc      4               ;sub-band 0
        dc      4               ;sub-band 1
```


BAD ORIGINAL

```
         dc       3                ;sub-band 2
         dc       3                ;sub-band 3
         dc       3                ;sub-band 4
         dc       3                ;sub-band 5
         dc       3                ;sub-band 6
         dc       3                ;sub-band 7
;end table 3-B.2c
         dc       3                ;sub-band 8
         dc       3                ;sub-band 9
         dc       3                ;sub-band 10
         dc       3                ;sub-band 11
;end table 3-B.2d
         dc       3                ;sub-band 12
         dc       3                ;sub-band 13
         dc       3                ;sub-band 14
         dc       3                ;sub-band 15
         dc       3                ;sub-band 16
         dc       3                ;sub-band 17
         dc       3                ;sub-band 18
         dc       3                ;sub-band 19
         dc       3                ;sub-band 20
         dc       3                ;sub-band 21
         dc       3                ;sub-band 22
         dc       3                ;sub-band 23
         dc       3                ;sub-band 24
         dc       3                ;sub-band 25
         dc       3                ;sub-band 26
         dc       3                ;sub-band 27
         dc       3                ;sub-band 28
         dc       3                ;sub-band 29
         dc       3                ;sub-band 30
         dc       3                ;sub-band 31

; Low sampling rates

skftbl_3
         dc       4                ;sub-band 0
         dc       4                ;sub-band 1
         dc       4                ;sub-band 2
         dc       4                ;sub-band 3

         dc       3                ;sub-band 4
         dc       3                ;sub-band 5
         dc       3                ;sub-band 6
         dc       3                ;sub-band 7
         dc       3                ;sub-band 8
         dc       3                ;sub-band 9
         dc       3                ;sub-band 10

         dc       2                ;sub-band 11
         dc       2                ;sub-band 12
         dc       2                ;sub-band 13
         dc       2                ;sub-band 14
         dc       2                ;sub-band 15
         dc       2                ;sub-band 16
         dc       2                ;sub-band 17
         dc       2                ;sub-band 18
         dc       2                ;sub-band 19
         dc       2                ;sub-band 20
         dc       2                ;sub-band 21
```

```
        dc      2                       ;sub-band 22
        dc      2                       ;sub-band 23
        dc      2                       ;sub-band 24
        dc      2                       ;sub-band 25
        dc      2                       ;sub-band 26
        dc      2                       ;sub-band 27
        dc      2                       ;sub-band 28
        dc      2                       ;sub-band 29
;end table 3-B.1
        dc      2                       ;sub-band 30
        dc      2                       ;sub-band 31

endsetbal_yhe
        endsec

        org     phe:

setbal
        move    y:skftbl,r1             ;get selected # of bits table address
        move    #NUMSUBBANDS,n0         ;access the right channel SBIndx
        bclr    #JOINT_at_SB_BOUND,y:<stereo    ;clear for initial sub-bands
        move    y:<sibound,r3           ;intensity stereo sub-band counter
        move    x:crcbits,r2            ;get CRC-16 bit counter

        dc      y:<maxsubs,_setb_40     ;output for applicable MAXSUBBANDS
        move    y:(r1)+,n4              ;get # of bits to use for this sub-band
        move    n4,n2                   ;to accumulate CRC-16 bits
        move    x:(r0),y0               ;get left channel SubBandIndex[SubBand]
        jsr     setvalue                ;output the left channel value
        move    (r2)+n2                 ;count bits covered by CRC-16 rtn

; if a mono type of frame, skip the right channel

        jset    #STEREO_vs_MONO,y:<stereo,_setb_30

; if not doing a joint stereo frame, handle the right channel

        jclr    #JOINT_FRAMING,y:<stereo,_setb_20

; if doing a joint stereo framing and frame upgraded to FULL stereo,
;       handle the right channel

        jset    #JOINT_at_FULL,y:<stereo,_setb_20

; if joint stereo has reached the sub-band boundary, skip the right channel

        jset    #JOINT_at_SB_BOUND,y:<stereo,_setb_30

; check if the sub-band intensity boundary has been reached

        jsr     chkjoint

; if joint stereo has reached the sub-band boundary, skip the right channel

        jset    #JOINT_at_SB_BOUND,y:<stereo,_setb_30

_setb_20
        move    x:(r0+n0),y0            ;get right channel SubBandIndex[SubBand]
        jsr     setvalue                ;output the right channel value
        move    (r2)+n2                 ;count bits covered by CRC-16 rtn
```

```
_setb_30
        move      r0 -                       ;next used sub-band

_setb_40
        move      r0,x:crcbits               ;store updated CRC-16 bit counter

        rts
```

```
        opt     fc
;
;  c 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  \MCODE\setancda.asm
; This routine outputs the ancillary data bytes to the output stream.
; on entry
;       r6 = current offset in output array
;       y:dataoptr = address in data byte input buffer to start from
;       y:bytecnt = count of bytes in input buffer not yet framed
;       y:maxbytes = max bytes per frame at given baud rate
; on exit
;       a = destroyed
;       b = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r0 = destroyed
;       r1 = destroyed
;       r4 = destroyed
;       n4 = destroyed

        include 'def.asm'
        include '..\common\ioequ.asm'
        include 'box_ctl.asm'

        section bytebuffer
        xdef    databytes

        org     yhe:
stsetancda_bytes

databytes       ds      DATABUFLEN              ;buffer for bytes received

endsetancda_bytes
        endsec

        section highmisc
        xdef    anctype
        xdef    baudrte                 ;data baud rate code from switches
        xdef    dataiptr
        xdef    dataoptr
        xdef    bytecnt
        xdef    bytesfrm
        xdef    maxbytes
        xdef    ancbits
        xdef    padbits

        org     yhe:
stsetancda_yhe

anctype         ds      1       ;type of count field after audio data:
                                ;       0 = 3 bit padded byte count
                                ;       1 = 8 bit data byte count
baudrte         ds      1       ;data baud rate code from switches
dataiptr        ds      1       ;ptr for next byte received
dataoptr        ds      1       ;ptr for next byte to insert into frame
bytecnt         ds      1       ;count of bytes yet to be output to frame
```

```
bytesfrm        ds      1       ;count of bytes for output to current frame
maxbytes        ds      1       ;max bytes that can go at baud rate
ancbits         ds      1       ;bits in current frame for ancillary data
padbits         ds      1       ;unallocated audio bits to set pad byte count

endsetancda_yhe
        endsec

        org     phe:

setancdata

;if not ancillary data byte count,
;   insert the count of pad bytes into the frame

        move    #anctype,r4             ;to check for data byte count type
        move    y:padbits,b             ;count of unallocated bits
        jclr    #0,y:(r4),_ancd_00      ;if not data byte count, do pad byte cnt

;insert the count of ancillary data bytes rather than
;  the CCS cdq standard count of pad bytes

        move    #BITSPERBYTE,n4         ;8 bits for byte count
        move    y:bytesfrm,y0           ;count of ancillary data bytes
        jmp     _ancd_05                ;insert the byte count

_ancd_00

;normal CCS cdq's encode the byte count of unallocated MUSICAM bits
;  divide number of unallocated bits by 8 (bits per byte) to get
;    truncated count of total bytes padded with 0

        lsr     b       #BITSFORPADDING,n4  ;divide by 2
                                        ; & set number of bits for pad count
        lsr     b                       ;divide by 2 again (==> by 4)
        lsr     b       #0,x0           ;divide by 2 again (==> by 8)
                                        ; & get set to zero count

        tst     b                       ;should never be negative
        tlt     x0,b                    ;if negative, set to zero
        move    b1,y0                   ;set up to insert pad count

_ancd_05

;encode the padded byte count or ancillary data byte count

        jsr     setvalue                ;insert byte count for decoder

        move    y:bytesfrm,b            ;if count of data bytes is zero
        tst     b                       ;test if no bytes this frame
        jeq     _ancd_100               ;no data bytes to insert

;now insert the bytes into current frame

        move    y:dataoptr,r5           ;address of next byte to be output
        move    #DATABUFLEN-1,m5        ;circular buffer
        move    #BITSPERBYTE,n4         ;number of bits to insert in the frame
        do      b,_ancd_10              ;output the number of bytes
        move    y:(r5)+,y0              ;word with the byte to insert
        jsr     setvalue                ;format the byte in the frame
```

```
        nop
_ancd_10
;temporarily disable data received interrupt to decrement unframed byte count

        bclr    #M_RIE,x:<<M_SCR

;while waiting for disable interrupt to take effect:

        move    r5,y:dataoptr            ;save addr of next byte for next frame
        move    #-1,m5                   ;uncircular buffer
        move    y:bytesfrm,y0            ;count of data bytes just framed

;interrupt should be cleared by now to safely get byte count maintained by
;       interrupt routine

        move    y:bytecnt,a              ;get latest byte count of unframed bytes
        sub     y0,a     #0,y0           ;subtract count of bytes just framed
                                         ; & get set to zero count
        tlt     y0,a                     ;if negative, zero count
;       tst     a                        ;make sure we're not negative
;       jge     _ancd_20                 ;if 0 or more, continue
;       clr     a                        ;reset to zero (just a precaution)
;
;_ancd_20
        move    a,y:bytecnt              ;save new unsent byte count

;turn the receive byte interrupt back on

        bset    #M_RIE,x:<<M_SCR         ;renable receive interrupt

_ancd_100

;pad 0 bits to the end of the audio portion of the frame

        move    y:audendpos,r0           ;get bit count to end of MUSICAM frame

;set flag for reed solomon (if reed solomon, skip the frame flush)

        move    #reedsolomon,r4          ;addr of the flag

        jsr     flushframe               ;pad frame with zeroes to MUSICAM end

        tst     a                        ;see if an overshoot ?????
        jlt     _ANCD_HELP               ;OVERSHOOT ERROR!!!!!!
        jmp     _ancd_150                ;OK, see if any client trailing bits

;;;pad 0 bits to the end of the audio portion of the frame
;;
;;      move    #0,y0                    ;init with zeros to pad last word
;;      move    y:audendw,x1             ;address of end of audio portion
;;      move    r6,b                     ;next o/p addr of current frame
;;      cmp     x1,b     #>24,a          ;if addresses eq, handle last few bits
;;                                       ; & set up for the next test
;;      jeq     _ancd_130                ;we're at the last word of audio
;;
;;;output last partially formatted data word before zero fill remainder of frame
;;
;;      move    y:<sc,x0                 ;get number of bits in last word
```

```
;;      sub     x0,a     #>24,x0       ;get number of bits left
;;      cmp     x0,a     #0,x0         ;24 - number of bits left
;;      jeq     _ancd_110              ;not partially formatted (y:sc == 0)
;;
;;      move    y:<curwd,b             ;get current output word
;;      rep     a                      ;output the necessary # of bits
;;      lsl     b
;;
;;      move    b1,y:(r6)+             ;save in the output
;;      move    x0,y:<sc               ;zero the current bit offset
;;
;;_ancd_110
;;
;;      clr     a                      ;output zero for remainder of frame
;;
;;_ancd_120
;;
;;;see if the last word of the audio portion of frame is to be output next
;;
;;      move    r6,b                   ;next o/p address of current frame
;;      cmp     x1,b                   ;see if last word next
;;      jeq     _ancd_130              ;last word, chk for any remaining bits
;;      move    a1,y:(r6)+             ;output frame word and incrment addr
;;      jmp     _ancd_120              ;continue to flush the buffer
;;
;;_ancd_130
;;
;;;handle the last word of the frame
;;
;;      move    y:audendb,b            ;bit offset signaling end of audio
;;      move    y:<sc,y1               ;get current formatted word offset
;;      sub     y1,b                   ;sub to get # bits remaining
;;;     tst     b                      ;test if any zero bits to output
;;      jeq     _ancd_150              ;if none, we're done
;;      jgt     _ancd_140              ;OK, output value

_ANCD_HELP

;ERROR!!! this case should not occur

        ON_BITALLOC_LED_CD             ;!!! error we've overshot

;!!!debug: dump the frame in question (pull of the ';' from next line)

;       jsr     dumpdata

;;      jmp     _ancd_150
;;
;;_ancd_140
;;      move    b,n4                   ;number of bits to output
;;      jsr     setvalue               ;pad word with zeroes as needed
;;
_ancd_150

;insert any client trailing bits

        INSERT_CLIENT_TRAILING_BITS_CD

        rts
```


BAD ORIGINAL

```
        opt fc
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \XCODE\scirec.asm

        title 'SCI receive ancillary data interrupt handler'

        include 'def.asm'
        include '..\common\ioequ.asm'

;these save variables for exclusive use by the scirec interrupt handlers only

        section highmisc
        xdef    scirecR7Save
        xdef    scirecN7Save
        xdef    scirecM7Save

        org     xhe:
stscirec_xhe

scirecR7Save    ds      1
scirecN7Save    ds      1
scirecM7Save    ds      1

endscirec_xhe
        endsec

; SCI xcode receive ancillary data interrupt

        org     pli:

scirec
        move    r7,x:scirecR7Save
        move    m7,x:scirecM7Save

        move    y:dataiptr,r7           ;get input data byte buffer pointer
        move    #DATABUFLEN-1,m7        ;circular buffer
        nop
        movep   x:<<M_SRXL,y:(r7)+      ;get the byte and store in buffer
        move    r7,y:dataiptr           ;update input data byte buffer pointer
        move    y:bytecnt,r7            ;increment the data byte counter
        move    #-1,m7                  ;no circular buffer ctl for count
        nop
        move    (r7)+                   ;increment
        move    r7,y:bytecnt            ;save the new byte count

        move    x:scirecM7Save,m7
        move    x:scirecR7Save,r7

_sci_90
        rti

;SCI xcode receive ancillary data interrupt exception

scirece
        move    r7,x:scirecR7Save

        movep   x:<<M_SSR,r7            ;clear the exception
        movep   x:<<M_SRXL,r7           ;eat the byte
```

```
move    x:scirecR7Save,r7
rti
```

```
        nolist
;
;  c  1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UXCODE\quantize.mac

QUANTIZE macro

; This routine is used to quantize the data.
; The resulting data is right justified in the result register.

;1st test to see if we are doing a Joint stereo quantize and if so,
; do the joint quantize routine and return the result from that routine;

        jclr    #JOINT_at_SB_BOUND,y:<stereo,_quant_10

        move    y0,a                                    ;get value to test register
        move    y:MaxiFact,y0                           ;get the Maxi scale factor
        tst     a               #lshftbl,r4             ;see if dividend is negative
        jlt     _jquan_10                               ;it is

; - dividend and - divisor

        move    y:(r4+n4),y1
        and     #$fe,ccr                                ;clear the carry bit
        rep     n4                                      ;value/scalefactor
        div     y0,a
        div     y0,a                                    ;one more div
        div     y0,a                                    ;one more div

        move    a0,y0                                   ;get result to a reg
        mpy     y0,y1,a         #qstbl,r4               ;left justify
        jmp     _jquan_20

; - divedend and + divisor

_jquan_10
        neg     a               y:(r4+n4),y1            ;make +
        and     #$fe,ccr                                ;clear the carry bit
        rep     n4                                      ;value/scalefactor
        div     y0,a
        div     y0,a                                    ;one more div
        div     y0,a                                    ;one more div

        move    a0,y0                                   ;get result to a reg
        mpy     -y0,y1,a        #qstbl,r4               ;left justify

_jquan_20
        move    a0,a
        tfr     x1,a            a,y0
        mac     x0,y0,a         y:(r4+n4),y1            ;form quantized result
        asr     a               y:<bitscnt,r4           ;divide by 2
                                                        ; & get bits used so far
        move    a,y0
        mpy     y1,y0,a         y:<sc,y1                ;right justify the bits
                                                        ; & = of bits left in curr word

;done with joint quantizing, go to the end of the macro

        jmp     _quant_900
```


BAD ORIGINAL

```
_quant_90

; This routine assumes that it must multiply two numbers together.
; One number is called P and the other number is called Q.
; P is unsigned and consists of an integer part (24 bits) and a
; fractional part (24 bits).
; Q is a signed fractional number (24 bits).
;
; P is of the form P1.P0 and Q is of the form .Q0 .
; The produce of P * Q is always less than 1.
;
; To perform the multiplication,
;
;        P1.P0
;   *      .Q0
;   ---------
;        .P0Q0
;      P1.Q0

; To do this in the dsp, assume the following register usage
;
;        P1 = y1
;        P0 = y1
;        Q0 = y0

;        the result (to 24 bits) is in a (as a signed value)


        move     x:(r5+n5),y1                        ;get P0
;3/24/94         move    a,y0                                  ;get Q0 in right registe
        mpy      y1,y0,a                             ;P1 * Q0 -
                                                     ; rslt will always be in a0
        asr      a                 y:(r5+n5),y1      ;adjust for integer * fractional

        move     a0,a                                ;move to right position
                                                     ; in accumulator
        macr     y1,y0,a           #qstbl,r4         ;P0 * Q0

        tfr      x1,a              a,y0
        mac      x0,y0,a           y:(r4+n4),y1      ;form quantized result
        asr      a                 y:<bitscnt,r4     ;divide by 2
                                                     ; & get bits used so far
        move     a,y0
        mpy      y1,y0,a           y:<sc,y1          ;right justify the bits
                                                     ; & # of bits left in curr word

_quant_900

        endm
        list
```


BAD ORIGINAL

```
        opt     fc,mex
;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \UXCODE\qcalcglo.asm  uses lower.asm and upper.asm - Larry values)

        title   'Calculate Global Masking Threshold'

; This routine is used to calculate the global masking threshold.

; on entry
;       r4 = address of masker structure (l memory)
;       r1 = address of GlobalMaskingThreshold (in slb's) (x memory)
;       x:<nmasker = number of maskers

; on exit
;       a = destroyed
;       b = destroyed
;       x0 = destroyed
;       x1 = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r2 = destroyed (pfmap)
;       r3 = destroyed (lmskr)
;       r4 = destroyed (rmskr)
;       r5 = destroyed (b_i)
;       r6 = destroyed
;       n1 = destroyed (thrsld - current index into Threshld
;       n2 = destroyed (mskrnum)
;       n4 = destroyed
;       n5 = destroyed (k)

        include 'def.asm'
        include '..\x1psycho\lower.asm'
        include '..\x1psycho\upper.asm'
        include '..\uxcode\dbadd.mac'

        org     pli:
QCalcGlo
; note: r4 is now free and could be used for Threshld

        move    #0,n2                           ;set to working on first mskr

        move    y:thresslb,n1                   ;start of threshold array (SLB)

; Find first masker which is not deleted.

        move    #>DELETEDMSKR,x0                ;deleted type
        move    #MASKERSTYPE,n4                 ;offset to type

        move    x:<nmasker,b                    ;get number of maskers
        tst     b         y:fmap,r2             ;and check for non zero
                                                ; & pfmap = fmap
        jeq     <_calc_10

        do      b,_calc_10
        move    x:(r4+n4),a                     ;get type
        cmp     x0,a      #MASKERSSIZE,n4       ;check if deleted
        jeq     <_calc_05
```


BAD ORIGINAL

```
        enddo                                   ;found a non-deleted masker
        jmp     <_calc_10

_calc_15
        move    (r4)+n4                         ;index to next masker
        move    #MASKERSTYPE,n4                 ;offset to masker type
        nop

_calc_10
        do      y:<nmskfreqs,_calc_90

        move    n1,r5                           ;get address of next quiet pwr
        move    y:(r2)+,n5                      ;k = *pfmap++
        move    #MASKERSBFREQ,n4                ;offset to BFreq
        move    x:(r5+n5),a                     ;get the quiet power in SLB's
        move    y:b_i,r5                        ;get base address of b_i table
        move    a,x:(r1)                        ;save as power in SLB's

        move    y:(r5+n5),y0                    ;BFreq = b_i
        move    y:(r4+n4),a                     ;rmskr->BFreq
        cmp     y0,a      #MASKERSTYPE,n4       ;rmskr->BFreq - BFreq
        jgt     <_calc_30

        move    #MASKERSSIZE,n4                 ;size of the structure
        move    r4,r3                           ;lmskr = rmskr
        move    (r4)+n4                         ;++rmskr

; Find next masker which is not deleted.

        move    #>DELETEDMSKR,x0                ;deleted type

        move    x:<nmasker,b                    ;get number of maskers
        tst     b         #MASKERSTYPE,n4       ;and check for non zero
                                                ; & offset to type
        jeq     <_calc_20

        do      b,_calc_20
        move    x:(r4+n4),a                     ;get type
        cmp     x0,a      #MASKERSSIZE,n4       ;check if deleted
        jeq     <_calc_25

        enddo                                   ;found a non-deleted masker
        jmp     <_calc_20

_calc_25
        move    (r4)+n4                         ;index to next masker
        move    #MASKERSTYPE,n4                 ;offset to masker type
        nop

_calc_20
        move    #MASKERSTYPE,n4
        move    #>1,n2                          ;set to not the first masker

_calc_30
        move    x:(r4+n4),a                     ;rmskr->Type
        move    #>ENDMSKR,x0                    ;end type
        cmp     x0,a      #MASKERSBFREQ,n4
        jeq     <_calc_40                       ;if at end don't process right

        move    y:(r4+n4),b                     ;rmskr->BFreq
```

```
        sub     y0,b     #.09375,x0           ;sdbark - rmskr->BFreq - BFreq
        cmp     x0,b     #MASKERSPWRDB,n4     ;sdbark - .09375
        jgt     <_calc_40                     ;check range

        move    #.03125,x1
        move    x:(r4+n4),y1                  ;rmskr->PowerDB

        LOWER_SLOPE

        move    y:(r4+n4),a                   ;rmskr->PowerDB
        sub     x1,a     x:(r1),x0            ;form masking skirt
                                              ; and get GlobalMasking Threshld
        DBADD
        move    b,x:(r1)

        move    y:(r5+n5),y0                  ;BFreq = b_i

_calc_40
        move    n2,a
        tst     a        #MASKERSBFREQ,n3
        jeq     <_calc_50

        move    y:(r3+n3),b                   ;lmskr->BFreq
        sub     y0,b     #.25,x0              ;lmskr->BFreq - BFreq
        neg     b        #MASKERSPWRDB,n3     ;BFreq - lmskr->BFreq
        cmp     x0,b     #.03125,x1
        jgt     <_calc_50

        move    x:(r3+n3),y1                  ;get the ->lmskr->PowerDb

        UPPER_SLOPE

        move    y:(r3+n3),a                   ;lmskr->ReducedPowerDb
        sub     x1,a     x:(r1),x0            ;form masking skirt
                                              ; & get GlobalMaskingThreshld
        DBADD
        move    b,x:(r1)

_calc_50
        move    (r1)+

_calc_90
        rts
```


BAD ORIGINAL

```
        opt     cex
;
;  (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UXCODE\findtona.asm

        title   'find tonals'

; This function is used to find the tonals. Once the tonal is found, it is
; replaced by a single power value which is the sum of 3 points.

; on entry
;       r1 = address of the power array (l memory)
;       r2 = address of the tonal structure (l memory)
;       r4 = address of the range table (y memory)

; on exit
;       r3 = # of tonals found
;
;       a = destroyed
;       b = destroyed
;       x0 = destroyed
;       x1 = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r1 = destroyed
;       r2 = destroyed
;       r5 = destroyed
;       n1 = destroyed
;       n2 = destroyed
;       n4 = destroyed

        include 'def.asm'

        org     phe:

findtona

        move    r1,r5                   ;save starting address

;       First compute the ending address

;       move    #>509,y1
        move    #>320,y1
        move    r1,a
        add     y1,a
        move    a,y1                    ;save ending address for later

        move    (r1)+                   ;start at power + 2
        move    (r1)+
        move    #0,r3                   ;ntonals = 0
        move    #-1,n1

;       This is the big loop where we look for tonals

_find_00

;       look for a local maximum

        move    l:(r1),b                ;pow[i]
```

```
        move    l:(r1-n1),a              ;pow[i-1]
        cmp     a,b      #1,n1           ;pow[i] - pow[i-1]
        jle     _find_39
        move    l:(r1+n1),a              ;pow[i-1]
        cmp     a,b                      ;pow[i] - pow[i+1]
        jle     _find_39

;       scale pow[i] down by 7.2 dB (should be 7.0 dB)

;       asr     b                        ;get 3/16 of power
;       asr     b
;       tfr     b,a                      ;move entire register
;       asr     b
;       asr     b
;       sub     b,a                      ;power is down 7.2 dB

;       scale pow[i] down by 6.0 db (ISO says 7.0 dB)

        asr     b                        ;get 1/4 of power
        asr     b
        tfr     b,a                      ;power is down by 6.0 db

;       Now search on each side to see if a tonal.

;       first determine search range.

        move    r1,b                     ;get the current index of pow
        move    r5,x1                    ;get starting position
        sub     x1,b     #2,y0           ;compute distance into array
        move    b1,x1                    ;move to right register
        mpy     x1,y0,b #>1,x0           ;shift right 6 bits
        move    b1,n4                    ;get offset into range table
        move    r1,n2                    ;save starting r1 value
        move    y:(r4+n4),b              ;get range
        sub     x0,b     (r1)-           ;range - 1
        move    b,x0                     ;save range - 1

;       search the lower side
;       must back off two from center address. one backoff was done above.

        move    (r1)-                    ;set r1 to starting value

        move    l:(r1)-,b                ;get first power value
        do      x0,_find_40              ;search range

;       cmp     b,a                      ;3/16 * pow[i] - pow[i-j]
        cmp     b,a                      ;1/4 * pow[i] - pow[i-j]
        jge     _find_30                 ;so far so good

        enddo
        move    n2,r1                    ;restore r1
        jmp     _find_39

_find_30
        move    l:(r1)-,b                ;get next power value
_find_40

;       now search the upper side

        move    n2,r1                    ;restore r1
```


BAD ORIGINAL

```
        nop
        move    r1 -                    ;set r1 to starting value
        move    r1,-

        move    l:(r1)-,b               ;get first power value
        do      x0,_find_42             ;search range

        cmp     b,a                     ;3/16 * pow[i] - pow[i-j]
        jge     _find_32                ;so far so good

        enddo
        move    n2,r1                   ;restore r1
        jmp     _find_39

_find_32
        move    l:(r1)-,b               ;get next power value
_find_42

        move    n2,r1                   ;restore r1

;       now we save the bin number in the tonal structure

        move    #TONALSBIN,n2           ;get bin offset
        move    #-1,n1                  ;set index
        move    x1,x:(r2+n2)            ;save the fft bin number

;       we found a local maximum and it was a tonal
;       add power of 3 hightst points

        move    l:(r1),b                ;pow[i]
        move    l:(r1+n1),a             ;pow[i-1]
        add     b,a     #1,n1           ;pow[i+1]+pow[i]
        move    #TONALSPWRDB,n2         ;get offset to power
        move    l:(r1+n1),b             ;pow[i-1]
        add     b,a     x0,b            ;pow[i+1]+pow[i]+pow[i-1]
        move    a,l:(r2+n2)             ;save in tonal array

;*************************************************************************
;       Now advance the r1 position to next possible position.
;       The next possible position is the current position + range+1.
;       We only advance it by range since the +1 is done at the bottom
;       of the loop.

;       move    r1,x0                   ;get  range
;       add     x0,b                    ;r1 + range
;       move    b1,r1
;*************************************************************************

;       10-8-91
;       Now advance the r1 position to next possible position.
;       The next possible position is the current position - 2 + 1.
;       We only advance it by 2 since the -1 is done at the bottom
;       of the loop.
;       This advancement is less than the old method because the old
;       method skipped over tonals which were higher and the the skipped
;       tonal was then considered as noise and generated a higher
;       masking threshold.  This caused less bits to be allocated to
;       the sub-band then there should have been.
;       Remember that the energy in a tonal is the sum of the power in
```


BAD ORIGINAL

```
;       the highest point and the left and right hand points
;       around the highest point.

        move    r1)-
        move    r1)-

; We come here when we have finished processing a tonal and put it in
; the tonal structure.

        move    #TONALSSIZE,n2          ;get size of tonal structure
        move    r3)-                    ;ntonals--
        move    r2)-n2                  ;advance to next entry

_find_39
        move    r1)-                    ;start looking at next point

        move    r1,a                    ;get current count
        cmp     y1,a    #-1,n1          ;get maximum count
        jle     _find_00

        rts

        end
```

```
     opt  fc
;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights
reserved.
;
; UXCODE\bitsallo.asm

     title     'Initialize bit output'

;routines:
;  setframelen: Sampling Rate 44100 (& sampling at 32000 for 399
kbs)
;         This routine handles the test for whether frames
;         need to be padded and set the working length (y:bitsfrm)
;         for the next frame as it performs the necessary ISO
formula
;         updates for the next frame. A padded frame length is
;         y:frmbits plus 8 bits;
;         Other Sampling Rates requitre no padding. In this case
;         the working frame bit length (y:bitsfrm) is set equal
;         to y:frmbits.
;
;

     include   'def.asm'
     include   'box_ctl.asm'

     org  phe:

setframelen

;set the working frame length in bits for the current frame to be
coded:
;if the frame requires no padding (most cases), y:<bitsfrm =
y:<frmbits
;determine if the frame is to be padded:
;  get frame's unpadded bit count
;  get current REST value (if not negative, no padding)
;   initialize as no padding in this frame (set code for frame
header)
;  get the DIFF value at the sampling rate and framing bit rate

     move y:frmbits,b         ;unpadded frame bit length
     move y:padrest,a         ;REST after last frame
     move #0,y1               ;indicate no padding
     tst  a    y:usediff,x0   ;see if padding needed,
                         ; & get the DIFF value
     jge  _padd_00       ;if not neg, no padding

;this frame is padded, add the number of bits as per ISO to normal
frame length
; and set the indication for the frame header taht the frame is
padded
```

```
; add the sampling frequency to REST as part of calculation for the
next frame

     move #>PAD_SLOT,x1         ;padded bits added to frame
     add  x1,b y:padrate,y0     ;add to unpadded frame bit length
                             ; & get the sampling rate value
     add  y0,a #>1,y1           ;add sampling rate to REST
                             ;set padded indication for frame header

_padd_00

;decrement the REST variable by the DIFF value for the next frame

     sub  x0,a               ;sub the DIFF value from REST
     move a,y:padrest           ;save update REST value for next
frame

;indicate if padded or not as determined above (for frame header)
;and set the frame in bit length

     move y1,y:usediff          ;indicate if padded or not
     move b,y:<bitsfrm          ;set bits in the frame
     rts

;bitpool()
;    This subroutine determines the number of bits available based
;    on the output bit rate and the type of framing
;
;***************************************************************
***************
;The table below is based on a Sampling Rate at 48,000 /sec and
shows
;the breakdown of bit counts based on bit rate o/p and choice of
frame type
;              Mono           Full          <------------ Joint Stereo
------------->
;kb  frame                   Stereo       4-bound      8-bound     12-bound
 16-bound
;rate      bits fix avail   fix avail    fix avail    fix avail    fix
avail  fix avail
;----      ----  --- -----  --- -----    --- -----    --- -----    ---
-----  --- -----
;384 9216    136  9080  224  8992  152  9064  168  9048  183  9033
195  9021
;256 6144     .   6008   .   5920   .   5992   .   5976   .   5961
 .   5949
;192 4608     .   4472   .   4384   .   4456   .   4440   .   4425
 .   4413
;128 3072     .   2936   .   2848   .   2920   .   2904   .   2889
 .   2877
;112 2688     .   2552   .   2464   .   2536   .   2520   .   2505
 .   2493
; 96 2304     .   2168   .   2080   .   2152   .   2136   .   2121
 .   2109
```

```
; 64 1536     .    1400    .    1312    .    1384    .    1368    .    1353
 .    1341
; 56 1344    136  1208  224  1120  152  1192  168  1176  183  1161
195  1149
;**************************************************************
***************/
;
;     y:<sibound = for joint stereo this is the sub-band boundary
;                below which sub-bands are full stereo
;                otherwise,
;                     only one channel (the left) is accounted for
;     y:<stereo = flags:
;                    bit 0 means stereo vs mono framing
;                               0 = stereo framing
;                     1 = mono framing
;               bit 2 is to simply indicate that joint stereo applies
;                               0 = NOT joint stereo framing type
;                               1 = IS joint stereo framing type
;               bit 3 is to indicate the full stereo upgrade by
allocate rtn
;                if joint stereo applies
;                               0 = normal joint stereo allocation
;                               1 = FULL STEREO allocation
;               bit 4 is to simply indicate the stereo intensity
sub-band
;                boundary has been reached if joint stereo applies
;                               0  =  NO  sub-bands  still  below
intensity boundary
;                               1  =  sub-bands  above  intensity
boundary
;               bit 11 does dual line transmission apply requiring
that a
;                block sequence number be appended to the coded
frame
;                               0 = dual line block sequence does NOT
apply
;                               1  =  dual  line  block  sequence
numbering APPLIES
;               bit 18 indicates whether or the crc checksum applies
;                     0 = NO do not account for checksum
;                     1 = YES do account for checksum
;
;     y:<maxsubs = maximum sub-bands at sampling rate, bit rate & 1
vs 2 chans
;     y:<bitsfrm = the total number of bits in a frame at the
specified
;               bit rate  if applicable, padded frame bits were
added
;                          to y:<frmbits)
;   these are used to determine if the frame requires a pad of 8
bits
;     y:padrate = sample rate value
;     y:padrest = updated REST value in ISO calculation.
;     y:usediff = DIFF value in ISO calculation  after determinating
```

```
;           whether padding is necessary, this variable is changed:
;                 0 = NOT a padded frame
;                 1 = frame was padded
;
; on exit:
;     x0 destroyed = returned number of required (fixed) bits
;     x1 destroyed = returned number of bits available for bit
allocation
;
;     a  destroyed
;     b  destroyed
;     r0 destroyed
;     r1 destroyed
;     r3 destroyed
;     r4 destroyed
;

     section   lowmisc
     xdef sc,curwd,bitsfrm,bitscnt

     org  yli:
stbitsallo_yli

sc   ds   1                  ;shift count
curwd     ds   1                  ;current word
bitsfrm   ds   1                  ;bit length of the current frame
bitscnt   ds   1                  ;count bits inserted in frame

endbitsallo_yli
     endsec

     org  phe:

bitpool

;Select the proper Allowed table:
; ISO:
;     1. for low sampling rates (24 or 16 K),
;          set ISO Extention Allowed table (Allowed_3)
;     2. for high sampling rates (48, 44.1 or 32 K):
;          a. based on MAXSUBBANDS less than 27,
;               set ISO lower bit rate Allowed table (Allowed_2)
;          b. else,
;               set ISO higher bit rate Allowed table (Allowed_1)
; CCS:
;     set ISO higher bit rate Allowed table (Allowed_1)

;low sampling rate:
;     test the frame header ID bit (if 0, it's a low sampling rate
frame)

     move #smplidbit,r0       ;addr of frame header ID bit (0 =
low)
     nop                 ;                              (1 = high)
```


BAD ORIGINAL

```
     ;set #0,y: r0),_bitp_030_A     ;if high rate, select Allowed
table

     move #Allowed_3,r0          ;addr of low sampling allowed table
     move #skftbl_3,r1           ;addr of the BAL bits table
     move #>15,x1                ;maximum position Allowed_3 table
     jmp  _bitp_010_A            ;go to store Allowed table address

_bitp_030_A

;high sampling rate:
; set the proper Allowed table address based on working MAXSUBBANDS
 y:<maxubs)
; if less than 27, used table 2

     move y:<maxsubs,x0          ;get current MAXSUBBANDS
     move #>27,a                 ;to see which of 2 tables applies
     move #>17,x1                ;maximum position Allowed_1 table
     move #skftbl_1,r1           ;addr of the BAL bits table
     cmp  x0,a #Allowed_1,r0     ;see if need the low bit rate table
                               ; & set up as Allowed_1 table
     jle  _bitp_010_A            ;Allowed_1 table applies

;select the lower bit rate Allowed table

     move #Allowed_2,r0
     move #skftbl_2,r1           ;addr of the BAL bits table
     move #>16,x1                ;maximum position Allowed_2 table

_bitp_010_A

;set the address of the selected Allowed table
;set the address of the selected BAL's bit table
;set the maximum position code

     move r0,y:<AllwAdd
     move r1,y:skftbl
     move x1,y:MaxPos

;determine the bits required for ancillary data (taken from audio
bit pool):
; start with bits required to store the padded data byte count in
frame

     move #anctype,r4            ;to see if data byte count applies
     move #>BITSFORPADDING,b     ;bits in the padded byte count

;if data byte count applies, change padded bits byte count  3 bits

;  to count (8 bits) of ancillary data bytes encoded in the frame

     jclr #0,y:(r4),_bitp_00   ;if not data byte, proceed
     move #>BITSPERBYTE,b          ;size of the ancillary data byte
count
```


BAD ORIGINAL

```
_bitp_00
     move y:maxbytes,y1          ;get max bytes at baud rate
     move y:bytecnt,a            ;get current count of bytes received
     cmp  y1,a #>BITSPERBYTE,x1    ;see max versus current count
                          ; & set multiplier
     jge  _bitp_05           ;if more than max, can only send max
     move a,y1               ;less than max, send all received

_bitp_05

;multiply the bytecount for bits per byte

     mpy  x1,y1,a               ;to get the required bit
     asr  a     y1,y:bytesfrm   ;shift integer result
                          ; & set byte count for framing
     move a0,a
     add  a,b               ;add bits to bits in byte count field

;!!!test
;!!!tst   move y:<bitsfrm,b    ;!!!test: get total frame bits
;!!!tst   lsr  b     #0,x1     ;!!!test: take half for ancillary
data
;!!!tst   lsr  b     #0,x1     ;!!!test: take quater for ancillary
data
;!!!tst   move x1,y:bytesfrm   ;!!!test: zero byte count for frame
;!!!test
     move b,y:ancbits           ;set ancillary data bit count

;set the number of fixed bits used, and the number of available
bits for audio

     clr  a     #0,x1           ;0 a as accum, zero CRC checksum bit
cnt

;set the fixed bits for the audio frame

     move #>NSYNC,x0            ;number of SYNC bits
     add  x0,a #>NSYST,x0       ;plus number of bits in frame system
hdr
     add  x0,a y:skftbl,r0      ;get base of used bits table

     jclr #PROTECT,y:<stereo,_bitp_35 ;skip checksum bits if no
protect
     move #>NCRCBITS,x1         ;add applicable bits for the checksum

_bitp_35
     add  x1,a               ;add checksum protection, if any

;in case of joint stereo, set the intensity sub-band boundary value

     move y:<sibound,r3

;accummulate the bit allocation bits for standard number of
```

```
sub-bands
; included in the frame for the left and right (if applicable)

     do    y:<maxsubs,_bitp_50

;always accumulate for the left channel

     move y:(r0)+,x1
     add  x1,a

;if doing one channel only, skip the right channel

     jset #STEREO_vs_MONO,y:<stereo,_bitp_40

;if NOT doing joint stereo framing or framing at FULL stereo,
;    add for the right channel

     jclr #JOINT_FRAMING,y:<stereo,_bitp_30
     jset #JOINT_at_FULL,y:<stereo,_bitp_30

;if doing Joint and we have reached the intensity sub-band
boundary,
;    skip the right channel

     jset #JOINT_at_SB_BOUND,y:<stereo,_bitp_40

;if doing Joint check to see if we have reached the intensity
sub-band boundary,
;    and if we just did, skip the right channel

     jsr  chkjoint
     jset #JOINT_at_SB_BOUND,y:<stereo,_bitp_40

_bitp_30

;stereo, add for the right channel

     add  x1,a

_bitp_40
     nop

_bitp_50
     move a,x0            ;return fixed bits
     move y:<bitsfrm,b        ;total size of frame in bits
     move b,y:frmendpos       ;bits to end of total frame
     move b,y:audendpos       ;bits to end of MUSICAM frame
     move b,y:bsnendpos       ;bits to end location for block seq
num
     move b,y:reedendpos      ;bits  to  end  total  frame   reed
solomon;

;if doing a split mode transmission, subtract the bit for the block
sequence
```

```
     jclr #SPLIT_MODE,y:<stereo,_bitp_70
     move #>BLOCK_SEQ_NUM_BITS,x1
     sub  x1,b
     move b,y:bsnendpos          ;bits to end location for block seq
num

_bitp_60

;if formatting a MONO split frame, divide the formatted frame bits
in half

     jclr #SPLIT_MONO_FRAME,y:<stereo,_bitp_70  ;if NOT, continue

;***** (start)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******
;***** (start)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******

;   divide the formatted frame bits in half

     lsr  b
     move b1,b

;***** (end)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******
;***** (end)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******

_bitp_70

;if reed solomon frames, subtract the reserved bits from the bit
pool

     move #reedsolomon,r4           ;addr of the flag
     nop
     jclr #0,y:(r4),_bitp_75  ;if no reed solomon bits, continue
     move y:trailbits,y1      ;get bits required for this frame
     sub  y1,b            ;reduce the bit pool

_bitp_75

;save end bit poisition of the MUSICAM frame
;    plus any bits required for client trailing bits and scale
factor crc's

     move b,y:frmendpos        ;bits to the end encoded frame

;subtract any trailing bits required by the client
;  and, if applicable, subtract bits for the next frame's scale
factor checksums

     move #private,r0          ;to see if skf crc's apply
     move #>CLIENT_TRAILING_BITS,y0  ;get count of client reserved
bits
```

```
     sub  y0,b =>NSKFCRCBITS*NUMSKFCKSUMS,y0 ;sub client bits
                         ; & set count of skf checksum bits
     ;clr #0,x::r0:,_bitp_80  ;if not appl, do not sub skf crc bits
     sub  y0,b            ;sub bits for skf checksum bits

_bitp_80

;set bit count to the end of the MUSICAM frame

     move b,y:audendpos        ;end MUSICAM up to client bits & skf
crc

;subtract the bits required for ancillary data

     move y:ancbits,y1         ;get count of ancillary data bits
     sub  y1,b            ;less the ancillary data bits

;subtract the accumulated frame fixed bits

     sub  a,b             ;total bits - fixed bits

;this leaves the bits available for allocation

     move b,x1            ;return number of audio data bits avail

;done in all cases with end bit positions set

     rts

;bitsallo()
;    This subroutine starts the bit allocation of values into the
;    frame buffer values are inserted by setvalue() and by
bitfree() below

; on entry
;    r6 = address of the output buffer
;    m6 = circular buffer control for OutData (2 frames 2*frame
wds)

; on exit
;    y:sc = 0
;    y:curwd = initialized (0) 1st word in frame buffer
;    r6 = address of the output buffer
;    m6 = circular buffer control for OutData (2 frames 2*frame
wds)
;
;    a = destroyed

bitsallo
     clr  a
     move a,y:<sc              ;initialize the shift count
     move a,y:<curwd           ;initialize curwd (1st bit in op
frame)
     move a,y:<bitscnt         ;start the bit counter of framed bits
```


BAD ORIGINAL

```
        rts

;bitsfree:
;       This routine flushes the last bits to the output buffer
;
; on entry
;       r6 = address of next word the output frame buffer (y memory)
;       y:<stereo bit 11 does dual line transmission apply requiring
that a
;                       block sequence number be appended to the coded
frame
;                                       0 = dual line block sequence does NOT
apply
;                                       1   =   dual   line   block   sequence
numbering APPLIES

; on exit
;       a = destroted
;       b = destroyed
;       x0 = destroyed
;       x1 = destroyed
;       y0 = destroyed
;       y1 = destroyed

        section   blkseqnums
        xdef seqnums
        xdef nxtseq,seqnum
        xdef frmendpos ;bit position of the true end of the frame
        xdef audendpos ;bit position of end of MUSICAM frame
        xdef bsnendpos ;bit position for block sequence number
        xdef spltrte
        xdef spltbnd
        xdef spltmaxsubs
        xdef spltpaddiff

;***** (start)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
 *******

        xdef nxtsc
        xdef nxtcurwd
        xdef nxtstrt

;***** (end)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
 *******

        org  yhe:
stbitsallo_yhe

;define the low order 12 bits of the coded block sequence numbers
for
;       dual output line bit allocation:
;       bits 0-9 contain the echoed block sequence number in the
;               range of 10 thru 31 (each bit is duplicated
;               for the even/odd line transmission)
```


BAD ORIGINAL

```
;     bit 10 = 1 for output over one of the 2 lines
;     bit 11 = 0 for output over the other of the 2 lines

seqnums
        dc     $000400          ;01 - 00 00 00 00 00 (sequence no. 0 =
00000)
        dc     $000403          ;01 - 00 00 00 00 11 (sequence no. 1 =
00001)
        dc     $00040c          ;01 - 00 00 00 11 00 (sequence no. 2 =
00010)
        dc     $00040f          ;01 - 00 00 00 11 11 (sequence no. 3 =
00011)
        dc     $000430          ;01 - 00 00 11 00 00 (sequence no. 4 =
00100)
        dc     $000433          ;01 - 00 00 11 00 11 (sequence no. 5 =
00101)
        dc     $00043c          ;01 - 00 00 11 11 00 (sequence no. 6 =
00110)
        dc     $00043f          ;01 - 00 00 11 11 11 (sequence no. 7 =
00111)
        dc     $0004c0          ;01 - 00 11 00 00 00 (sequence no. 8 =
01000)
        dc     $0004c3          ;01 - 00 11 00 00 11 (sequence no. 9 =
01001)
        dc     $0004cc          ;01 - 00 11 00 11 00 (sequence no. 10 =
01010)
        dc     $0004cf          ;01 - 00 11 00 11 11 (sequence no. 11 =
01011)
        dc     $0004f0          ;01 - 00 11 11 00 00 (sequence no. 12 =
01100)
        dc     $0004f3          ;01 - 00 11 11 00 11 (sequence no. 13 =
01101)
        dc     $0004fc          ;01 - 00 11 11 11 00 (sequence no. 14 =
01110)
        dc     $0004ff          ;01 - 00 11 11 11 11 (sequence no. 15 =
01111)
        dc     $000700          ;01 - 11 00 00 00 00 (sequence no. 16 =
10000)
        dc     $000703          ;01 - 11 00 00 00 11 (sequence no. 17 =
10001)
        dc     $00070c          ;01 - 11 00 00 11 00 (sequence no. 18 =
10010)
        dc     $00070f          ;01 - 11 00 00 11 11 (sequence no. 19 =
10011)
        dc     $000730          ;01 - 11 00 11 00 00 (sequence no. 20 =
10100)
        dc     $000733          ;01 - 11 00 11 00 11 (sequence no. 21 =
10101)
        dc     $00073c          ;01 - 11 00 11 11 00 (sequence no. 22 =
10110)
        dc     $00073f          ;01 - 11 00 11 11 11 (sequence no. 23 =
10111)
        dc     $0007c0          ;01 - 11 11 00 00 00 (sequence no. 24 =
11000)
```

```
        dc    $0007e3          ;01 - 11 11 00 00 11  sequence no. 25 =
11001)
        dc    $0007ec          ;01 - 11 11 00 11 00 .sequence no. 26 =
11010)
        dc    $0007ef          ;01 - 11 11 00 11 11 .sequence no. 27 =
11011)
        dc    $0007f0          ;01 - 11 11 11 00 00 (sequence no. 28 =
11100)
        dc    $0007f3          ;01 - 11 11 11 00 11 (sequence no. 29 =
11101)
        dc    $0007fc          ;01 - 11 11 11 11 00 (sequence no. 30 =
11110)
        dc    $0007ff          ;01 - 11 11 11 11 11  sequence no. 31 =
11111)
endsequence
nxtseq          ds    1     ;address of next
seqnum          ds    1     ;block sequence number to set A-bit
frmendpos ds    1    ;bit position of the true end of the frame
audendpos ds    1    ;bit position of end of MUSICAM frame
bsnendpos ds    1    ;bit position for block sequence number
spltrte   ds    1    ;split mono frame bit rate code for frame hdr
spltbnd   ds    1    ;split mono frame bit rate code for bandwdith
spltmaxsubs     ds    1     ;split mono frame MAXSUBBANDS
spltpaddiff     ds    1     ;frame padding calc: DIFF @ sample/bit
rates

;***** (start)  SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******

nxtsc           ds    1     ;y:<sc value to start next frame
nxtcurwd  ds    1    ;y:<curwd partly formatted word-start next
frame
nxtstrt         ds    1     ;y:<frmstrt buffer address to start next
frame

;***** (end)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******

endbitsallo_yhe
     endsec

bitsfree

;pad 0 bits thru the end of the coded frame including any client
trailing bits

     move y:frmendpos,r0        ;bit count thru CLIENT bits

;set flag for reed solomon (if reed solomon, skip the frame flush)

     move #reedsolomon,r4           ;addr of the flag

     jsr  flushframe            ;pad frame with zeroes
```


BAD ORIGINAL

```
        tst   a                 ;check for an overshoot?????
        jge   _free_00          ;OK, see if we have a split frame

;OVERSHOOT ERROR!!! this case should not occur

        ON_BITALLOC_LED_CD          ;!!! error we've overshot

;!!!debug: dump the frame in question (pull of the ';' from next
line)

;       jsr   dumpdata

        jmp   _free_90          ;done with bad frame

_free_00

;see if split frame applies, if not, we should have coded all bits
in frame

        jclr #SPLIT_MODE,y:<stereo,_free_90      ;if not split, chk end
of frame

;if NOT a split mono frame, output the block sequence number

        jclr #SPLIT_MONO_FRAME,y:<stereo,_free_20

;***** (start)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******
;***** (start)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******

;format the block sequence number for the last word of the frame
buffer

        move y:nxtseq,r1            ;block sequence number to output
        move #31,m1                 ;circular buffer thru blk seq num tbl
        nop
        move y:(r1)+,x1             ;get blk seq num, incr for next frame
;!!!dbg    move y:(r1),x1  ;!!!dbg keep the same BSN at end of frame
        move x1,y:seqnum
        move r1,y:nxtseq            ;save for next frame blk seq num

;test if one of the receiving lines is down and this frame is a
split frame

        TST_CLR_TRAN_A_ERROR_CD,_spit_00    ;NOT A-bit set to 1
        bset #A_BIT_OFFSET_ODD,y:seqnum     ;set bit for line 1
        bset #A_BIT_OFFSET_EVEN,y:seqnum    ;set bit for line 1

_spit_00

;     determine word and bit offsets for the end of the entire frame

        move y:<outsize,m0          ;set for circular buffer control
```

```
     move y:<frmsc,x0          ;set frame start address bit offset
     move #>24,a               ;set number bits in a word
     move #>24,y1              ;set number bits in a word
     sub  x0,a y:<bitsfrm,b    ;set bit count for frame in 1st word
                          ; & get bit count for current frame
     cmp  y1,a y:<frmstrt,r0   ;see if entire 1st word of frame
                          ; & set frame start address
     jeq  _splt_10        ;if word fits, go right into loop

;only part of 1st word contains the frame,
; a. sub bits from entire frame bit count
; b. increment address counter

     sub  a,b  (r0)+

_splt_10

;adjust address to end of the frame as per 24 bits per word giving
the
;word address and bit count to start the next frame

     cmp  y1,b            ;see if reached last word
     jlt  _splt_20        ;if so, set eoframe word & bit offsets
     sub  y1,b (r0)+
     jmp  _splt_10

_splt_20
     move b,y:nxtsc       ;bit offset start next formatted frame
     tst  b     r0,y:nxtstrt   ;if bit offset not zero, next addr
set
                          ; & set buffer addr start next frame
     jne  _splt_25        ;if offset 0, incr addr start next frame
     move (r0)-                ;back up addr to end of current frame

_splt_25

;set up the end word of the current frame with the
;  left justified block sequence number
;the end bits in the frame will shift the end word back right

     bclr #4,y:<not_appl       ;indicate end word of frame NOT done
     move y:seqnum,b           ;get the block sequence number

     move #24-BLOCK_SEQ_NUM_BITS,r2 ;set num bsn bits to roll left

;left justify the block sequence number

     do   r2,_splt_30
     rol  b                ;roll left up to 1st data bit

_splt_30

;position at the end of the formatted frame and the end of the
frame buffer
```

```
; prior to the block sequence number

     move y:<outsize,m1          ;set circular buffer ctl for source
     move y:<sc,a                ;numb partial formatted source bits
     move #0,r3                  ;no bits to rotate from source yet
     tst  a     r6,r1            ;see if any bits partially formatted
                            ; & set the source start address
     jeq  _splt_40          ;no partial bits, start at last insert

;the end of the frame is partially formatted in y:curwd for y:sc
bits

     move y:<sc,r3          ;set bit counter in partial format word
     move y:<curwd,a             ;get right justified part formatted
word

_splt_40

;see if source is ready to get the previous word

     move a1,b0                  ;save current shifted word
     move r3,a              ;get the bit counter
     tst  a     b0,a1            ;test for zero & restore shifted word
     jne  _splt_50          ;is still bits to go, continue

;test if we just finished the 1st word in the source and if so,
; we're done, output the 1st word of the frame and continue

     move y:<frmstrt,x1          ;backed to the start of the frame?
     move r1,a              ;last word addr eq to frame start addr
     cmp  x1,a (r1)-             ;test equal, & back up to previous
word
     jeq  _splt_120         ;if eq, we're done

     move #24,r3                 ;start with a new word

;see if this new word to be processed is the frame start word.
; if so, adjust for the bit offset to the start of the frame SYNC

     move r1,a              ;see if new word addr eq to frame start
     cmp  x1,a #>24,a            ;test if at the 1st word of frame
                            ; & set for bit count if it is 1st word
     jne  _splt_45          ;if not 1st word, get the new word
     move y:<frmsc,x1            ;get frame start address bit offset
     sub  x1,a              ;calculate bits in the 1st frame word
     move a,r3              ;and move to the source word bit ctl

_splt_45

;take the next word to be processed

     move y:(r1),a          ;get previous word

_splt_50
```


BAD ORIGINAL

```
     move  r3--                    ;decrement shifted bit
     ror   a             ;move source bit to carry bit
     jcs   _splt_60      ;see if carry bit is a 1
     bclr #10,y:<not_appl          ;flag that carry bit is 0
     jmp   _splt_70

_splt_60
     bset #10,y:<not_appl          ;flag that carry bit is 1

_splt_70

;output the carry bit twice for line 1 and line 2

     do    #2,_splt_110

;see if destination is ready to get the previous word

     move b1,a0                    ;save current shifted word
     move r2,b           ;get the bit counter
     tst   b     a0,b1             ;test for zero & restore shifted word
     jne   _splt_80      ;is still bits to go, continue

;see if this is the end word of the frame and if so,
; set the y:nxtcurwd for the start of the next frame
; and make any adjustments for the current frame

     jset #4,y:<not_appl,_splt_78
     bset #4,y:<not_appl           ;indicate end word handled
     move b1,y:nxtcurwd            ;start of the next frame formatted wd
     move y:nxtsc,b      ;get the start bit for the next frame
     tst   b     b,r2    ;see if zero
                         ; & set the value to roll left
     jeq   _splt_76      ;if zero, end word is all set

;get current buffer end word roll left to abut the previous frame
start bit
; with the end bit of current frame

     clr   b             ;zero the b register
     move y:(r0),b0      ;get end word from frame buffer
     do    r2,_splt_72             ;shift the end word bits in b0
     asl   b             ; so they are left justified

_splt_72

;roll the end word to isolate the end bits for the frame buffer

     move a0,b1                    ;restore formatted end word
     do    r2,_splt_74             ;shift the nxtcurwd bits into b1
     asr   b             ; so they are left justified

_splt_74

;store the reformatted end word (and start of previous frame)
```

```
;    and restore the formatted end word with r2 set according to
shift

     move b0,y:(r0)-            ;store formatted end word back in buf
                           ; & decrment address for next word o/p
     move a0,b1                 ;restore formatted end word
     jmp  _splt_80         ;continue by inserting bits

_splt_76

;no bits needed to be shifted, the end word is all set

     move a0,b1                 ;restore formatted end word

_splt_78

;store reformatted word in the frame buffer

     move #24,r2                ;start with a new word
     move b1,y:(r0)-            ;put new word out to buffer & back up

_splt_80

;either clear or set the carry bit

     jset #10,y:<not_appl,_splt_90  ;is carry bit is to be restored
to 1
     andi #$FE,ccr         ;set the carry bit to 0
     jmp  _splt_100

_splt_90
     ori  #$01,ccr         ;set the carry bit to 1

_splt_100

;count the bits inserted and insert the bit

     move (r2)-                 ;decrement shifted bit
     ror  b                ;move carry bit into word

_splt_110

;go back for the next bit from the source

     jmp  _splt_40

_splt_120

;see if partially formatted 1st word in the frame
; if so, right adjust the partial word and and it with end of
previous frame

     move #>24,a                ;set bits per word
     move r2,x0                 ;get shifted bit count downer
```


BAD ORIGINAL

```
     sub  x0,a              ;see if any bits shifted in to b1
     cmp  y1,a a,r2         ;test for no bits partially formatted
                            ; & set the shift bit counter
     jeq  _splt_140         ;if no bits to go, continue

     move y::r0!,a          ;get the word at frame start

;right align the last word in previous frame

     do   r2,_splt_130
     asr  a                 ;shift right up to 1st data bit

_splt_130

;now abut the frame start bits (a0) with the end bits of previous
frame (a1)

     move b1,a0                   ;partial formatted word to a0

;shift left to align the last word in previous frame
;  with start bits of current frame in a1

     do   r2,_splt_135
     asl  a                 ;shift left up to 1st data bit

_splt_135

;now put the reformatted word in the proper register

     move a1,b              ;a1 = end of prev start of current frame

_splt_140
     move b1,y:(r0)         ;output new 1st word out to buffer
     move #-1,m0                  ;reset to linear buffer control
     move #-1,m1                  ;reset to linear buffer control
     jmp  _free_90          ;set addr for skf checksums - next frame

;***** (end)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******
;***** (end)   SPLIT_MONO_FRAME ONLY (to make 2nd copy of frame):
  *******

;SPLIT_MODE frame but NOT a SPLIT_MONO_FRAME position for the BSN

_free_20
     move y:nxtseq,r1             ;block sequence number to output
     move #31,m1                  ;circular buffer thru blk seq num tbl
     move #>BLOCK_SEQ_NUM_BITS,x0  ;number of bits for block seq
num
     move y:(r1)-,x1              ;get blk seq num, incr for next frame
;!!!dbg    move y:(r1),x1  ;!!!dbg keep the same BSN at end of frame
     move x1,y:seqnum             ;store the selected sequence number
     move r1,y:nxtseq             ;save for next frame blk seq num
     move #-1,m1                  ;restore to linear buffer ctl
```

```
;test if one of the receiving lines is down and this frame is a
split frame

     TST_CLR_TRAN_A_ERROR_CD,_free_30    ;NOT A-bit set to 1
     bset #A_BIT_OFFSET_ODD,y:seqnum     ;set bit for line 1
     bset #A_BIT_OFFSET_EVEN,y:seqnum    ;set bit for line 2

_free_30
     move #BLOCK_SEQ_NUM_BITS,n4    ;number of bits for block seq
num
     move y:seqnum,y0          ;block seq number to output
     jsr  setvalue        ;add blk seq num in last word

_free_90

;set address for the next frame's scale factor checksums
; a. get bit count for CLIENT bits and the scale factor checksums
; b. get bit count to the end of the formatted frame (block seq
number)
; c. if this is a combined mode split mono frame, double the bit
count
;    for CLIENT and checksums
; d. determine word address and bit count to come back and insert
the
;    scale factor checksums in the already coded frame

     move #>CLIENT_TRAILING_BITS,r0
     move #>NSKFCRCBITS*NUMSKFCKSUMS,n0
     move y:bsnendpos,b          ;bit count to end of frame
     move (r0)+n0                ;bits for client + checksums
     move r0,n0                  ;if need to be doubled

;set flag for reed solomon (if reed solomon, scale factor crc next
to insert)

     move #reedsolomon,r4            ;addr of the flag

;test for a split mono frame in order to double the bit count

     jclr #SPLIT_MONO_FRAME,y:<stereo,_free_92     ;if not, continue
     move (r0)+n0                ;double bit count

_free_92

;for reed solomon, save current word address and bit offset for
;  the insertion of the next frame's scale factor checksums

     jclr #0,y:(r4),_free_93  ;if not reed solomon, continue
     move r6,x:skfcrcwd        ;word addr after anc data & client
bits
     move y:<sc,x0        ;get bit offset into next word to o/p
     move x0,x:skfcrcbt        ;bit offset after anc data & client
bits
```


BAD ORIGINAL

```
;now flush the rest of the frame with zero bits

     move #not_appl,r4          ;use this addr for the flush to be
done
     bclr #0,y:<not_appl        ;make sure bit is zero for flush
     move y:reedendpos,r1            ;bit count thru rest of the
buffer

;pad the remainder of the frame with zero bits

     jsr  flushframe            ;pad frame with zeroes
     tst  a                ;check for an overshoot?????
     jge  _free_97         ;OK, skip info for next frame skf crc's

;OVERSHOOT ERROR!!! this case should not occur

     ON_BITALLOC_LED_CD         ;!!! error we've overshot

;!!!debug: dump the frame in question (pull of the ';' from next
line)

;    jsr  dumpdata
     jmp  _free_97         ;skip info for next frame skf crc's

_free_93

;subtact the bits for client and checksums from the end of frame
bit count

     move r0,x0                 ;bit count client and checksums
     sub  x0,b y:<frmstrt,r0    ;sub from end frame bit count
                           ; & curr frame start address

;set start of frame address and circular buffer ctl in order to
;calculate address and bit offset to store the next frame's
checksums

;    move y:<frmstrt,r0         ;curr frame start address
     move y:<outsize,m0         ;circ buffer control

;get bits partially formatted in the 1st word
;account for the 1st partially formatted word of the frame

     move #>24,a                ;to determine bits in 1st word
     move y:<frmsc,x0           ;get bit offset start frame
     sub  x0,a #>24,x0          ;calc bits in 1st word of frame
     sub  a,b  (r0)+            ;sub 1st wd patrial bits
                           ; & increment the address

;loop subtracting 24 bits per word from end of frame bit count and
incrementing
; the address to reach the place for the next frame's scale factor
checksums
```


BAD ORIGINAL

```
_free_94
     cmp   x0,b              ;24 bits vs current bit counter
     jlt   _free_96          ;if less, we reached the address

;subtract 24 bits from end of frame bit counter

     sub   x0,b (r0)-            ;sub 24 bits from curr bit count
                             ; & increment the address
     jmp   _free_94          ;continue looping

_free_96

;save the calculated address and the bit offset to code the next
frame's crc's

     move r0,x:skfcrcwd          ;save address
     move b,x:skfcrcbt           ;save bit offset
     move #-1,m0                 ;restore linear buffer ctl

_free_97

;clear the flag that this frame is a split mono frame

     bclr #1,x:private

;if this is not split mono frame, go to validate the proper end of
frame

     jclr #SPLIT_MONO_FRAME,y:<stereo,_free_98

;set the flag that this frame is a split mono frame

     bset #1,x:private

;doing a split mono frame: set controls for starting the next frame

     move y:nxtsc,x0             ;set the y:<sc bit offset to start
     move x0,y:<sc           ;store bit offset to start y:<curwd
     move y:nxtcurwd,x0          ;get the y:<curwd formatted word
     move x0,y:<curwd            ;store 1st partial formatted word
     move y:nxtstrt,x0           ;get frame buffer start address
;!!!dbg   move x0,y:<frmstrt           ;store frame start address
     move x0,y:<frmnext          ;store frame start address
     jmp  _free_100          ;we're done

_free_98

;ensure that we have coded to the end of the frame

     move y:<bitsfrm,x0          ;get true frame length in bits
     move y:<bitscnt,a           ;get count of bits output so far
;!!!dbg   cmp  x0,a r6,y:<frmstrt  ;these should be equal
     cmp  x0,a r6,y:<frmnext  ;these should be equal
                             ; & save for start of next frame
```

```
        jeq   _free_100         ;OK!! all went according to plan

;FRAME ENCODE ERROR!!! this case should not occur

        ON_BITALLOC_LED_CL          ;!!! error we've overshot

;;!!debug: dump the frame in question (pull of the ';' from next
line)

;       jsr   dumpdata

        move #framebuf,r0           ;start pointers over
        move y:<outmus,n0           ;to advance 1 frame
;!!!dbg   move r0,y:<frmstrt              ;at beginning of the buffer
        move r0,y:<frmnext          ;at beginning of the buffer
        move (r0)+n0                ;address of the second frame
        move r0,y:<oprptr           ;output read pointer 1 frame ahaead

_free_100

        rts
```

```
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; UXCODE\fftr16b.asm
;
; Radix 2, In-Place, Decimation-In-Time FFT  fast .
;
; Last Update 18-Sept-90   Version 1.0
;
fftr16b  macro   points,data,coef,coef1,dacol
fftr16b  ident   1,0
;
; Radix 2 Decimation in Time In-Place Fast Fourier Transform Routine
;
;    Real input and complex output data
;        Real data in X memory
;        Imaginary data in Y memory
;    Normally ordered input data
;    Bit-reversed complex output data
;       Coefficient lookup table
;        -Cosine values in X memory
;        -Sine values in Y memory
;        -fast index search in X & Y memory
;
; Macro Call - fftr16b   points,data,coef,coef1,dacol
;
;       points      number of points (16-32768, power of 2)
;       data        start of data buffer
;       coef        start of sine/cosine table
;       dacol       start of index table
;
; Alters Data ALU Registers
;       x1      x0      y1      y0
;       a2      a1      a0      a
;       b2      b1      b0      b
;
; Alters Address Registers
;       r0      n0      m0
;       r1      n1      m1
;       r2      n2      m2
;       r3      n3
;       r4      n4      m4
;       r5      n5      m5
;       r6      n6      m6
;               n7
;
; Alters Program Control Registers
;       pc      sr
;
; Uses 6 locations on System Stack
;
; Latest Revision -  18-Sept-90
;
    move #data,r0              ;initialize input pointer
    move #points/4,n0          ;initialize input and output pointers offset
    move n0,n1                 ;initialize input pointers offset
    move n0,n4                 ;initialize output pointers offset
    move #1,n2                 ;initialize groups per pass
    move #coef,r4              ;initialize sine/cosine input pointers
    move #1024,n4                ;relative address
```


BAD ORIGINAL

```
     move #@cvi(@log(points)/@log(2)-3),r3
;
; Do first and second Radix 2 FFT passes, combined as 4-point butterflies
;
     lua   (r0)+n0,r1          ;initialize input B pointer
     move #points-1,m0         ;initialize address modifiers
     move m0,m1                ;for modulo addressing
     move m0,m4
     lua   (r1)+n1,r4          ;initialize output C pointer
     move m0,m5
     move x:(r0),x0
     lua   (r4)+n4,r5          ;initialize output D pointer
     do    n0,_twopass
     tfr   x0,a        x:(r1),y1
     tfr   y1,b        x:(r4),y0
     add   y0,a        x:(r5)-,x1                  ;ar+cr
     add   x1,b                                    ;br+dr
     add   a,b                                     ;ar'=(ar+cr)+(br+dr)
     subl  b,a         b,x:(r0)+                   ;br'=(ar+cr)-(br+dr)
     tfr   x0,a        a,x:(r1)+
     sub   y0,a        x1,b                        ;cr'=ar-cr
     sub   y1,b        a,x:(r4)                    ;ci'=dr-br
     move  x:(r0),x0   b,y:(r4)+
_twopass
     move  (r0)+n0
;
; Do the complex FFT using butterfly kernel to 2nd last pass
;
     do    #@cvi(@log(points)/@log(2)-3),_end
     move  r0,n3                  ;save the beginning address
     move  #dacol,r2              ;reset the index table
     move  n0,b1
     lsr   b
     move  b1,n0
     move  n0,n7                  ;save the input offset
     do    r3,_toendpass2
     move  r0,r4                  ;initialize output pointers
     lua   (r0)+n0,r1             ;initialize input B pointer
     move  n0,n1                  ;initialize all the input output offset
     move  n0,n4
     move  n0,n5
     lua   (r1)-,r5               ;initialize output D pointer
     do    n2,_endgroup                        ;calculate the group FFT
     move  y:(r2)+,r6
     move  x:(r5),a       y:(r0),b
     move  (r6)-n6
     move  x:(r1),x1      y:(r6),y0
     move  x:(r6),x0
     do    n0,_bufknl                          ;Kernel FFT processing
     mac   x1,y0,b        y:(r1)+,y1
     macr  -x0,y1,b       a,x:(r5)+       y:(r0),a
     subl  b,a            x:(r0),b        b,y:(r4)
     mac   -x1,x0,b       x:(r0)+,a       a,y:(r5)
     macr  -y1,y0,b       x:(r1),x1
     subl  b,a            b,x:(r4)+       y:(r0),b
_bufknl
     move  a,x:(r5)+n5    y:(r1)+n1,y1
     move  x:(r0)+n0,x1   y:(r4)+n4,y1
_endgroup
     move  n3,r0              ;reset the beginning address for next pass
```

```
    move n1,b1                        ;update the new group number
    lsr  b      n1,a1
    lsl  a      b1,n1
    move a1,n2
_endpass2
;
; Do last pass for all the complex FFTs
;
    lua  (r0)-,r1                     ;initialize input B pointer
    move #2,n0                        ;initialize FFT elements in each group
    move r0,r4                        ;initialize output C pointer
    move n0,n1                        ;initialize all input/output offset
    move n0,n4
    move n0,n5
    move r1,r5                        ;initialize output D pointer
    move y:(r2)-,r6
    move y:(r0),b
    move (r6)-n6
    do   n2,_endgroup1                ;each group is just one kernel process
    move x:(r1),x1      y:(r6),y0
    move x:(r6),x0
    mac  x1,y0,b        y:(r1)+n1,y1
    macr -x0,y1,b       y:(r0),a
    subl b,a            x:(r0),b          b,y:(r4)
    mac  -x1,x0,b       x:(r0)+n0,a       a,y:(r5)
    macr -y1,y0,b       y:(r2)-,r6
    subl b,a            b,x:(r4)+n4       y:(r0),b
    move a,x:(r5)+n5
    move (r6)-n6
_endgroup1
;
; Do the half upper real part's FFT
;
    move #data,r0
    move n7,n0
    move n0,n1
    move n0,n4
    lua  (r0)+n0,r1
    move #1,n2
    lua  (r1)-n1,r4
    move (r3)-
    lua  (r4)-n4,r5
    move x:(r0),a
    move x:(r1),y0
    do   n0,_uponep
    add  y0,a           a,b                          ;ar'=ar+cr
    subl a,b            a,x:(r0)-                    ;br'=ar-cr
    move b,x:(r1)+
    move x:(r5)-,b1
    neg  b              x:(r1),a                     ;ci'=-dr
    move b1,y:(r4)+
    move x:(r1),y0
_uponep
    move (r1)-n1
_end
;
; Do the beginning four point FFT at last pass
;
    move #data,r0
    move #2,n0
```


BAD ORIGINAL

```
      move #dacc1,r1
      move x:(r1)+,x0
      tfr  x0,a            x:(r1),y0
      sub  y0,a            y:(r1),r6          ;br'=ar-cr
      move a,x:(r0)+n0
      move x:(r1)+,b1
      neg  b                (r6)-n6           ;ci'=-dr
      move b1,y:(r0)+n0
;
; Complex 2 point FFT for last pass
;
      lua  (r0)+,r1
      move y:(r0),b
      move x:(r1),x1       y:(r6),y0
      move x:(r6)-,x0
      mac  x1,y0,b         y:(r1),y1
      macr -x0,y1,b        y:(r0),a      ;ci=ai+bicos-brsin
      subl b,a             b,y:(r0)      ;di=2ai-ci
      move x:(r0),b
      mac  -x1,x0,b        a,y:(r1)
      macr -y1,y0,b        x:(r0),a      ;cr=ar+brcos+bisin
      subl b,a             b,x:(r0)      ;dr=2ar-cr
      move a,x:(r1)

      endm
```

```
         opt      fc,cex,mex
 ;
 ;  (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
 ;
 ;  \UXCODE\setvalue.asm

 ; This routine is used to output bits to the output bit buffer.
 ; The basic idea is to look at 3 different cases:
 ;       1. The new bits fit entirely in the current word with room to spare.
 ;       2. The new bits fit exactly in the current word.
 ;       3. The new bits exceed the availiable room in the current word and
 ;          thus the current word is filled and a new word is started.

         title   'Set Value'

 ; on entry
 ;       r6 = address of the next word to the output buffer (y memory)
 ;       y0 = value to output (right justified)
 ;       n4 = number of bits to output (1-16)
 ;
 ;       y:<curwd = current word being formed for the frame
 ;       y:<sc = current bit position in current word being formed for the frame
 ;       y:<bitscnt = count of bits put to the frame
 ;
 ; on exit
 ;       a = destroyed
 ;       b = destroyed
 ;       y0 = destroyed
 ;       y1 = destroyed
 ;       r4 = destroyed
 ;
 ;       r6 = updated for next word in output buffer (OutData)
 ;
 ;       y:<curwd = updated with bit changes last inserted
 ;       y:<sc = updated bit position into the curren word being formated
 ;       y:<bitcnt = update count of bits put to the frame
 ;

        include '..\uxcode\setvalue.mac'

        section ytables
        xdef    shifttbl
        xdef    ldshftbl

        org     yhe:
stsetvalue_yhe

shifttbl
        dc      $000000                         ;place holder
        dc      $400000                         ;shift value for 1 bit
        dc      $200000                         ;shift value for 2 bit
        dc      $100000                         ;shift value for 3 bit
        dc      $080000                         ;shift value for 4 bit
        dc      $040000                         ;shift value for 5 bit
        dc      $020000                         ;shift value for 6 bit
        dc      $010000                         ;shift value for 7 bit
        dc      $008000                         ;shift value for 8 bit
        dc      $004000                         ;shift value for 9 bit
        dc      $002000                         ;shift value for 10 bit
        dc      $001000                         ;shift value for 11 bit
```

```
        dc      $000800                         ;shift value for 12 bit
        dc      $000400                         ;shift value for 13 bit
        dc      $000200                         ;shift value for 14 bit
        dc      $000100                         ;shift value for 15 bit
        dc      $000080                         ;shift value for 16 bit

ldshftbl
        dc      $000000                         ;place holder
        dc      $000001                         ;shift left 1 bit
        dc      $000002                         ;shift left 2 bits
        dc      $000004                         ;shift left 3 bits
        dc      $000008                         ;shift left 4 bits
        dc      $000010                         ;shift left 5 bits
        dc      $000020                         ;shift left 6 bits
        dc      $000040                         ;shift left 7 bits
        dc      $000080                         ;shift left 8 bits
        dc      $000100                         ;shift left 9 bits
        dc      $000200                         ;shift left 10 bits
        dc      $000400                         ;shift left 11 bits
        dc      $000800                         ;shift left 12 bits
        dc      $001000                         ;shift left 13 bits
        dc      $002000                         ;shift left 14 bits
        dc      $004000                         ;shift left 15 bits
        dc      $008000                         ;shift left 16 bits

endsetvalue_yhe
        endsec

        section highmisc
        xdef    svbl
        xdef    svn4

        org     xhe:
stsetvalue_xhe

svbl    ds      1
svn4    ds      1

endsetvalue_xhe
        endsec

        org     pli:

setvalue

;set up for the setvalue macro

        SETUP4SETVALUE
;       move    y:<sc,y1                ;get # of bits left in current word
;       move    n4,b                    ;set # of bits
;       clr     a       y:<bitscnt,r4   ;prepare a register
;                                       ; & get # of bits used so far
;       move    y0,a0                   ;put values into proper register

;use the setvalue macro

        SETVALUE

        rts
```

```
        nolist
;
;  c  1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  \DXCODE\setvalue.mac
;
; This routine is used to output bits to the output bit buffer.
; The basic idea is to look at 3 different cases:
;       1. The new bits fit entirely in the current word with room to spare.
;       2. The new bits fit exactly in the current word.
;       3. The new bits exceed the available room in the current word and
;          thus the current word is filled and a new word is started.
;
;       y:<curwd = current word being formed for the frame
;       y:<sc = current bit position in current word being formed for the frame
;       y:<bitscnt = count of bits put to the frame
;
; on entry
;       b  = number of bits to output (1-16, same as n4)
;       r6 = address of the next word to the output buffer (y memory)
;       a2 = 0
;       a1 = 0
;       a0 = value to output (right justified)
;       y1 = y:<sc (# of bits left in current word)
;       r4 = y:<bitscnt (number of bits output up to this call)
;       n4 = number of bits to output (1-16)
;
; on exit
;       a = destroyed
;       b = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r4 = destroyed
;
;       n4 = MUST BE SAFE ACROSS THIS CALL
;       x0 = MUST BE SAFE ACROSS THIS CALL
;       x1 = MUST BE SAFE ACROSS THIS CALL
;
;       r6 = updated for next word in output buffer (OutData)
;
;       y:<curwd = updated with bit changes last inserted
;       y:<sc = updated bit position into the curren word being formated
;       y:<bitscnt = update count of bits put to the frame
;

SETUP4SETVALUE  macro
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The next 4 lines should be in quantize.asm, setvalue.asm,...
; NOTE: quantize.mac already leaves the value in a0
; They should be removed from this routine.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        move    y:<sc,y1                        ;get # of bits left in current word
        move    n4,b                            ;set # of bits
        clr     a           y:<bitscnt,r4       ;get # of bits used so far
        move    y0,a0                           ;put value into proper register
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        endm

SETVALUE        macro
        add     y1,b        #>24,y1             ;add bits to o/p to offset
```


BAD ORIGINAL

```
                                        ; & set compare to 24 bits/word
        move    r4,-n4                  ;update total bits used so far

;see if this value will fit totally in current output word

        cmp     y1,b    r4,y:<bitscnt   ;see if new value fits
                                        ; & save new total bit count

        move    #ldshftbl,r4            ;get shift table address

        jlt     _setv_70                ;fits within current word
        jeq     _setv_60                ;exactly fits

; the current value is too big so we must do it in 2 parts.
;       part 1 - do the part which fits in the remaining bits.
;       part 2 - do the part which is left over.

; NOTE: b2 and b0 will be zero as a
; result of this operation

; find the number of bits left in the current word

;!!!N/A move    a0,y0                   ;save bits to output in a save register
        move    y:<sc,a                 ;get # of bits used in current word
        sub     y1,a    x0,x:svb1       ;get # of bits which just fit
                                        ;save x0 register
        neg     a       y:<curwd,x0     ;make -
                                        ; get current word we are working on

        move    n4,x:svn4               ;save the # of bits

        move    a,n4                    ;save # for this pass
        move    y0,y:<curwd             ;save as the new current word. Note that
                                        ;   we don't need to mask off the unused
                                        ;   upper bits since the word will be
                                        ;   shifted left soon.

; Move the current word left to make room for the new bits.
; The current word will be completely full after completing this section.

        move    y:(r4+n4),y1            ;get shift value
        mpy     y1,x0,a #>24,y1         ;shift old bits for new value

; Now move the msb's of the input right to fit into the lsb of the
; current word.

        sub     y1,b    x:svb1,x0       ;compute # of bits in next word
                                        ; & restore x0
        move    b,n4                    ;number of bits leftover
        move    #shifttbl,r4            ;address of right shift table
        move    a0,a                    ;move to correct register
        move    y:(r4+n4),y1            ;get shift value
        mac     y0,y1,a b,y:<sc         ;shift input word right into a1
                                        ; & insert new value at end of new curwd
        move    a1,y:(r6)+              ;output word to the buffer

        move    x:svn4,n4               ;restore the # of bits to output

        jmp     _setv_90                ;and we are done
```

```
; The current value just fits

_setv_60
        move    y:<curwd,y0             ;get current word in output buf

        move    y:(r4+n4),y1            ;get left shift value

        mac     y1,y0,a b0,y:<sc        ;shift old bits for new value
                                        ; & set bits used in current word to 0

        move    a0,y:(r6)+              ;output word to the buffer

        jmp     _setv_90

; this is the case when the value fits in the current word

_setv_70
        move    y:<curwd,y0             ;get current word in output buf

        move    y:(r4+n4),y1            ;get left shift value

        mac     y1,y0,a b,y:<sc         ;shift old bits for new value
                                        ; & update bits used in current word

        move    a0,y:<curwd             ;save current output word

_setv_90

        endm
        list
```

```
        opt     fc
;
;  c  1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
;  UKCODE\setcrc.asm

        title   'Set the checksum word'

; These routines maintain the checksum protection portion in the frame
;    clrcrc:  initializes the checksum portion in the frame by inserting 16
;        0 bits and thereby saving space for the calculated result
;        the 16-bit check sum after the header and before the bit allocations
;    info in bits 32-47 of the frame.
;    setcrc:  calls the routine to calculate the check sum and outputs
;        the 16-bit check sum after the header and before the bit allocations
;    info in bits 32-47 of the frame.

; on entry
;       r6 = current offset in output array
;       y:sc = shift count

; on exit
;       a = destroyed
;       b = destroyed
;       y0 = destroyed
;       y1 = destroyed
;       r4 = destroyed
;       n4 = destroyed
;

        include 'def.asm'

        section lowmisc
;;      xdef    frmaddr
        xdef    frmsc
        xdef    crcaddr
        xdef    crcsc

        org     yli:
stsetcrc_yli

;;frmaddr       ds      1               ;address of start of channel frame heade
frmsc   ds      1               ;bit offset into word to start channel frame
crcaddr ds      1               ;address of start of frame's CRC checksum
crcsc   ds      1               ;bit offset of start of frame's CRC checksum

endsetcrc_yli
        endsec

        section highmisc
        xdef    crcbits
        xdef    crcold
        xdef    chksum

        org     xhe:
stsetcrc_xhe

crcbits ds      1               ;NEW: accum span of bits for CRC-16 rtn
crcold  ds      1               ;OLD: fixed span of bits for CRC-16 rtn
chksum  ds      1               ;save calculated checksum
```


BAD ORIGINAL

```
endsetcrc_xhe
        endsec

        org     phe:

clrcrc

;this subroutine clears the checksum in the frame buffer
;  and saves its address in the frame buffer

        move    r6,y:<crcaddr             ;save address for inserting crc checksum
        move    y:<sc,x1                  ;current bit offset for CRC checksum
        clr     a          x1,y:<crcsc    ;zeroes for the checksum
                                          ; & save the CRC starting bit offset
        move    a,y0                      ;value to be output
        move    #NCRCBITS,n4              ;number of bits
        move    #CRC_BITS_A,r1            ;insert bit cnt for header & checksum
        move    r1,x:crcbits              ;init bit ctr for span covered by CRC-16
        jsr     setvalue                  ;output the value

        rts

setcrc

; x:crcbits = accumulator of bits covered by CRC-16 routine

;this subroutine calls the calculate checksum routine
;  and then inserts the result into frame buffer
; a. set starting address and bit offset of this channel frame header
; b. calculate the offset to start the checksum calculation

;;      move    y:<frmaddr,r0             ;get address of start of frame buffer
        move    y:<frmstrt,r0             ;get address of start of frame buffer
        move    m6,m0                     ;set circular buffer control
        move    y:<frmsc,a                ;get the starting bit offset of frame

        move    #>CRC_SUM_BIT_OFFSET,x1 ;calculate msb position from which to
                                          ;   start calculating the checksum
        add     x1,a     #>24,x1          ;set offset to start checksum calculate
                                          ; & to check overflow to next word
        cmp     x1,a                      ;see if offset to start in next word
        jlt     _scrc_a          ;if less, we're all set

;adjust address up 1 position and ajust bit offset to start for CRC-16 rtn

        sub     x1,a                      ;bits for 1 word to adjust bit offset
        move    (r0)+                     ;increment start word address

_scrc_a
        move    a,x1                      ;bit offset to start checksum calculate
        move    #>CRC_VALUE,y1            ;set the checksum divisor

;for ISO old or new CRC-16 controls:
; set the checksum seed value and the number of bits covered by the checksum

        jset    #CRC_OLD_vs_NEW,y:<stereo,_scrc_00

        move    x:crcold,r1               ;get OLD bit ctr for span over CRC-16
        move    #0,x0                     ;OLD: seed the checksum with 0's
```

```
        jmp     _scrc_10                    ;go to do the crc check

_scrc_00

;ISO new CRC_16 controls:

        move    x:crcbits,r1                ;get NEW bit ctr for span over CRC-16
        move    #$ffff00,x0                 ;NEW: seed the checksum with F's

_scrc_10
        jsr     crc                         ;do the checksum

;now, insert 16 bit checksum value

        move    y:<crcaddr,r0               ;address for start of the checksum
        move    a1,x:chksum                 ;save checksum returned from crc rtn
        clr     a         x:chksum,x1       ;set up to shift checksum
        move    x:chksum,a0                 ;set checksum in lower part of reg

;isolate the bits to shift for storing:
;  part in crcaddr and part in crcaddr + 1
;  or all in crcaddr

        move    #>24,b                      ;get bits in a word
        move    y:<crcsc,x0                 ;get bit offset to store CRC checksum
        sub     x0,b      #>NCRCBITS,x0     ;get bits remaining in word
                                            ; & get number of bits for CRC checksum
        cmp     x0,b      b1,y1             ;test if CRC wholly in one word
                                            ; & save number of bits for 1st shift
        jeq     _no_shift                   ;if equal, no shift
        jgt     _one_shift                  ;if more than enough room in word

;we have to do two shifts for overlapping 2 words
; 1. shift the checksum over two bytes to position for shift into a1

        do      #24-NCRCBITS,_shift_a
        asl     a

_shift_a

; 2. shift bits to offset into a1

        do      y1,_store_1st
        asl     a

_store_1st

; 3. store 1st portion from checksum into 1st word

        move    a1,x1                       ;bits for 1st word
        move    y:(r0),b                    ;get 1st word at that address
        or      x1,b                        ;set the low bits (were 0) to sum
        move    b1,y:(r0)+                  ;store back into the frame
                                            ; & increment for 2nd word
        jmp     _shift_2                    ;now store 2nd portion in 2nd word

_one_shift

;checksum fits within the 1st word
```


BAD ORIGINAL

```
        sub     x0,b                    ;calculate numb bits to shift
        do      b,_shift_1              ;shift up to offset for CRC
        asl     a

_shift_1

;store shifted checksum value

        move    a0,x1                   ;a0 now positioned

_no_shift

;last NCRCBITS at that address

        move    y:(r0),b                ;get the word at that address
        or      x1,b                    ;set the low 16 bits (were 0) to sum
        move    b1,y:(r0)               ;store back into the frame
        move    #-1,m0                  ;restore to linear buffer control

        rts
```

```
        opt     fc,mex
;
; (c) 1994. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \MUXCODE\setctls.asm

        title   'Encoder set transmission line controls'

; This routine is used to interpret the transmission line selection
;       and phase lock loop controls to set the variables required
;       for the bit allocation conditions and output line selection
;       and front panel leds.
;
;
; destroyed:
;       register a
;       register x0
;       register r0
;       register r1
;       register r2

        include 'def.asm'
        include 'box_ctl.asm'

        org     phe:

setctls

;initialize stero control settings to reflect current transmission

        bclr    #SPLIT_MODE,y:<stereo
        bclr    #SPLIT_MONO_FRAME,y:<stereo
        bclr    #NO_LINES,y:<stereo
        bclr    #BOTH_LINES,y:<stereo
        bclr    #SUMMARY_ALARM,y:<stereo

;check the selected transmission lines and the phase lock loops
        move    #select1,r0             ;addr of the line 1 select flag
        move    #select2,r1             ;addr of the line 2 select flag
        jset    #0,x:(r0),_ctls_10      ;if line 1 selected
        jset    #0,x:(r1),_ctls_20      ;if line 2 selected

;neither line selected

        bset    #NO_LINES,y:<stereo
        jmp     _ctls_20

_ctls_10

;line 1 selected, check if line 2 also selected
; and if so, indicate both lines selected

        jclr    #0,x:(r1),_ctls_20      ;if line 2 not selected

;both lines selected, set as redundant

        bset    #BOTH_LINES,y:<stereo

_ctls_20
```

```
;if the device supports it (DUAL_LINES in box_ctl.asm is set to 1),
; check if redundant should be set to split mode and whether bit allocation
; should account for block sequence numbering:
;       yes if (128 or 112) bit rate qualifies (#SPLIT_APPLIES)
;               AND neither line is specifically selected

        if DUAL_LINES==1

        jclr    #SPLIT_APPLIES,y:<stereo,_ctls_40
        jclr    #NO_LINES,y:<stereo,_ctls_40
        bset    #SPLIT_MODE,y:<stereo

; further, if the receiver has a problem with one line,
;       go into split MONO frame mode for frame 1.2 the normal size

        TST_CLR_REC_A_ERROR_CD,_ctls_40
        bset    #SPLIT_MONO_FRAME,y:<stereo

        endif

;indicate redundant mode, unless already set

_ctls_40
        rts
```

1. Introduction

1.1 cdqPRIMA Overview

The cdqPRIMA is an audio CODEC which is used to compress and decompress audio for transmission over a digital facility such as ISDN, T1, E1 and satellite. In addition to its audio compression capabilities, it has a rich set of monitor and control M&C features made possible by a powerful control processor and command language. These M&C capabilities provide the cdqPRIMA with unique capabilities not found in audio only CODEC's.

CDQPrima™ Technical Features

CDQPrima Model	110	120	210	220	230
Mechanical Features					
Dimensions: 19" Rack Mount	1U high	1U high	2U high	2U high	2U high
Digital Interface Module slots	1	1	3	3	3
World Power Supply, rear power switch	X	X	X	X	X
Dial and control keypad	X	X	X	X	X
Backlit LCD display	character	character	character	character	graphic
Digital LED average & peak VU meters		X		X	X
L/R correlation & stereo image display		X		X	X
Scrolling text messages on VU meters		X		X	X
Intelligent headphone monitor system		X		X	X

X = always present • = hardware/software option; for example, • 3 means optional 3

CDQPrima Model	110	120	210	220	230
Compression Algorithms					
CCS MUSICAM®	X	X	X	X	X
ISO/MPEG Layer II	X	X	X	X	X
CCITT G.722	X	X	X	X	X
16, 24, 32 & 48 kHz sampling rates	X	X	X	X	X
22.05 & 44.1 kHz sampling rates	•	•	•	•	•
Additional algorithm capacity	X	X	X	X	X

X = always present • = hardware/software option; for example, • 3 means optional 3

BAD ORIGINAL

CDQPrima Model	110	120	210	220	230
Audio I/O, SMPTE & Ancillary Data					
18-bit A/D and D/A converters	X	X	X	X	X
Gold plated Neutrik® XLR audio connectors	X	X	X	X	X
AES/EBU, S/PDIF	• DB9	DB9	XLR	XLR	XLR
Automatic rate adaptation	X	X	X	X	X
Optical Digital I/O			•	•	•
Spectrum analyzer & phase display					X
SMPTE Time Code			•	•	•
Asynchronous ancillary data	X	X	X	X	X
Synchronous ancillary data	•	•	•	•	•

X = always present • = hardware/software option; for example, • 3 means optional 3

CDQPrima Model	110	120	210	220	230
Command and Control					
68020 Integrated Support Processor	X	X	X	X	X
Software update via RS232 & inband ISDN	X	X	X	X	X
J.52 (H.221) BONDING	X	X	X	X	X
Extensive on-line help	X	X	X	X	X
Headphone select and level control keypad		X		X	X
4-button cue keypad		X		X	X
Hot keys & extended feature keypad					X
Full remote control via RS232 & RS485	X	X	X	X	X
Front panel RS232 remote control port		X		X	X
Optically isolated remote control inputs	• 4	• 4	• 8	• 8	• 8
Dry floating relay contacts or TTL outputs	• 4	• 4	• 8	• 8	• 8
Virtual control lines connecting each unit	12	12	12	12	12
RS232 control port, no modem control	X	X			
RS232 control port, full modem control			X	X	X
RS485 control port			X	X	X
Programmable summary alarm relay	X	X	X	X	X
Programmable silence detector		X		X	X
Programmable peak level detector		X		X	X
Bit error rate detector	X	X	X	X	X
Out-of-frame detector	X	X	X	X	X

X = always present • = hardware/software option; for example, • 3 means optional 3

CDQPrima Model	110	120	210	220	230
Additional Options Available					
ISDN/X.21/RS422/V.35 DIF modules	• 1	• 1	• 3	• 3	• 3
Windows® remote control software	•	•	•	•	•
Psychoacoustic parameter adjustment	•	•	•	•	•
ITU-T J.52 error protection	•	•	•	•	•
Analog stereo input limiter	•	•	•	•	•

X = always present • = hardware/software option; for example, • 3 means optional 3

The cdqPRIMA family falls into two broad categories, the 1xx and the 2xx families. The 1xx family is 1U (1.75") high hand hold 1 Digital Interface Module (DIM) while the 2xx family is 2U (3.5") high and holds 3 DIM's. Each DIM connects the cdqPRIMA to the digital transmission facility.

The block diagram if the cdqPRIMA is shown below.

cdqPRIMA

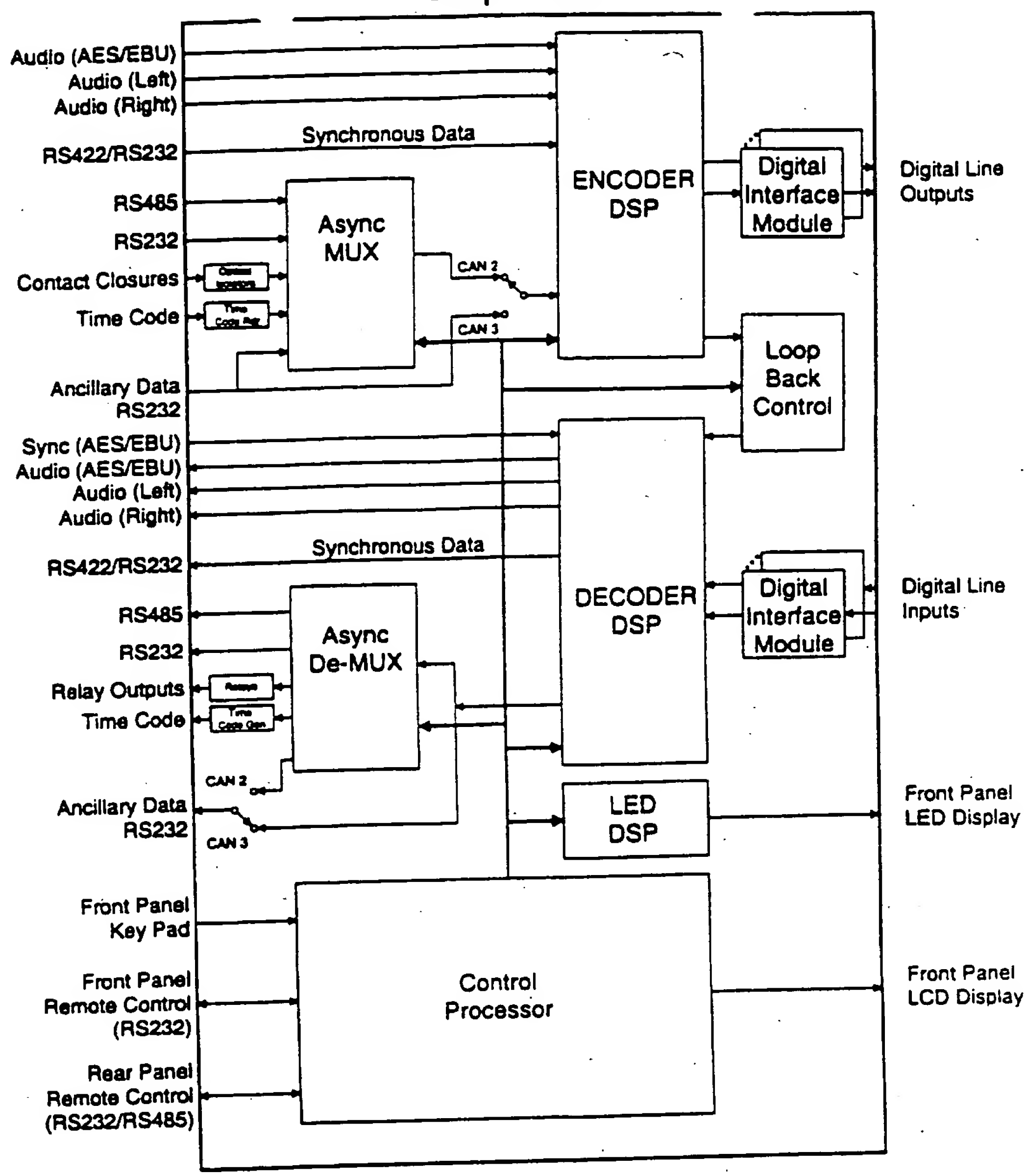


Figure 1
PRIMA High Level Block Diagram

Figure 1-1
cdqPRIMA High Level Block Diagram

1.1.1 Model 110 Description

The cdqPRIMA 110 provides a rich set of basic features which are included in the entire cdqPRIMA family such as a LCD display and control keypad in a 1U (1.75") 19" unit. The keypad allows operation of all the CODEC features. This unit also includes a rear panel remote control connector, 4 optical isolated monitor inputs and 4 control relay outputs. Ancillary data is also included in the cdqPRIMA 110. One Digital Interface Module (DIM) can used to interface with digital networks.

1.1.2 Model 120 Description

The cdqPRIMA 120 builds on the model 110 by adding a LED level display, front panel headphone output, front panel remote control and AES/EBU digital audio I/O. The AES/EBU digital audio interface on the 1xx series utilizes a DB9 connecter (an adaptor cable is available to convert from the DB9 to the standard XLR connectors). The keypad of the 120 includes buttons to control the headphone output source and level.

The LED level display provides a sophisticated level meter with peak hold, as well as stereo image and stereo corrolation capabilities. The LED level display can also display scrolling messages to the user. Such messages are helpful in alerting and cueing.

The 120 adds additional keys to the keypad for headphone control.

1.1.3 Model 210 Description

The features of the model 210 are identical the the 110 with several additional features. The 2xx series is housed in a 2U (3.50") by 19 inch enclosure and inclueds 8 optical isolators and 8 relays. SMPTE time code and optical digital audio are optionally available. Three Digital Interface Modules (DIM's) can used for interfacing to digital networks. On this model, the AES/EBU connectors are XLR instead of the DB9 on the 1xx series.

1.1.4 Model 220 Description

The features of the model 220 are identical to the 120 with the addition of 4 more optical isolators and 4 more relays.

1.1.5 Model 230 Description

This model provides all of the features of the 220 with the addition of a graphics display which can be used for measurements such as real time spectral analysis. The 230 also provides an enhanced keypad which adds measurement hot keys plus user programmable hot keys.

1.2 Digital Transmission Networks

1.2.1 Overview

The selection of the digital transmission facility must be considered when using the cdqPRIMA. The terrestrial network falls into two broad classification and these are the dedicated and switched networks. The dedicated network is, as the name implies, a dedicated path between two points. Examples of dedicated services are DDS56, T1 and E1. Typically, dedicated service is expensive but should be use if continuous connectivity is anticipated. If a dedicated or leased line is appropriate, it must have a CSU/DSU (Customer Service Unit / Data Service Unit) installed at each end. These units are responsible for converting the V.35 or X.21 signals into signals compatible with the network. They are relatively inexpensive and readily available from numerous manufactures and require no special instructions.

The digital switched network is attractive when occasional use is required because the cost of the service is computed based on a monthly fee plus the actual time the service is used. This is exactly like a conventional phone and the rates charged by the service providers are relative inexpensive and comparable to standard telephone rates.

Two examples of switched long distance terrestrial networks are the ATT ACCUNET Switched 56 network/ISDN and the Sprint VPN network. Both are digital networks and are candidates for use with the cdqPRIMA. The Sprint VPN network uses digital lines which were designed for speech and includes digital echo cancelers. The effect of these echo cancelers is to modify the digital bitstream in an attempt to remove what it thinks are echoes. This modification of the digital bit stream is disastrous to the cdqPRIMA because it expects the receiver to receive a binary 1 when it transmits a 1. Fortunately, the echo cancelers are easily disabled by using a proper CSU/DSU. In particular, a CSU/DSU must be equipped with an echo canceller disabler if it is to be used in the Sprint VPN network. This is a common option in switched CSU/DSU's and must be ordered if the long distance carrier is Sprint.

The ATT Accunet Switched 56 network or ISDN is intended for data and voice and does not require echo suppression facilities in the CSU/DSU.

There is another consideration when using the terrestrial switch 56 kb network and that is 4 wire verses 2 wire. In various regions of the United States, different regional operating companies use different technology to transmit the 56 kb data from the customer premise to the central office. The two technologies are called 2-wire and 4-wire. When ordering the local phone line (local loop), you must inquire about the circuit type - 2 wire or 4 wire and then order an appropriate CSU/DSU.

Satellite facilities require no special attention. Only a standard 56 kbps, 64 kbps, ... 384 kbs data line is required.

The cdqPRIMA is relatively immune to digital bit errors. If a binary 1 is occasionally changed to a 0 or visa versa, it has minimal impact. Synchronization is maintained even during error burst of up to .1 second. However, in either the satellite and terrestrial facilities.

a slip (the complete loss or addition of a bit) causes the receiver circuity to lose lock and then require framing. This entire process is statistical but usually only takes about .2 seconds. During this time, the receiver mutes and no audio is output.

1.2.2 Digital Transmission Facilities

1.2.2.1 Digital Data Service ("Nailed-Up" DDS)

This is the origional digital data service. It provides 56 kbs over a dedicated circuit. This technology is based on the telephone companies internal 64 kbs systems but 1 bit out of each 8 is robbed from the user for use by the telephone company to provide signalling information. This signalling information conveys such information such as dialing digits and on/off hook.

1.2.2.2 Switch 56

Switched 56 was the first switched digital technology transmission technology provided by the telephone companies. It utilizes the 56 kbs transport technology within the telco's as the DDS service described above.

1.2.2.3 The ISDN Basic Rate Interface (BRI)

ISDN is a new technology which is use to transport either 56 or 64 kbs. Utilizing ISDN, a single copper wire pair from the telephone company central office to the customer premis (a basic rate interface - BRI) can transport two B channels and one D channel. Each B channel can be either 56 or 64 kbs and the D channel transmits 16 kbs.

ISDN is computer to computer communication because it allows the central office computer to communicate with the customer premis computer. This customer premis computer is called a terminal adaptor (TA). This sophisticated computer to computer communication is accomplished over the D channel and does not rob any bits from either of the B channels. Since the central office computer is in contact with the customer premis computer, sophisticated communication is possible. For example, the central office computer can ask the customer primis computer if it will accept a data call at 64 kbs.

ISDN is the low bandwidth low cost interconnect method provided by the telephone companies. The rates of ISDN are similar to a normal analog telephone line.

1.2.2.4 Primary Rate ISDN (T1 & E1)

While ISDN provides 64 kbs service, T1 provides 24 64 kbs channels (1.544 mbs). E1 provides 32, 64 kbs channels. This increased bandwidth comes at an additional cost.

1.2.2.5 Long-Distance Interconnectivity (56-to-64kbps)

ISDN and switched 56 are available internationally. 56 kbs ISDN interconnects with switched 56 both nationally and internationally. This capability provides incredible world wide low cost connedtivity.

1.2.3 Other Digital transmission Paths

'While terrestrial facilities such as ISDN are popular, there are several other tecdhnologies for digital transmission which should be considered. RF transmission facilities form another class of transmission and are an alternative to terresterial transmission.

1.2.3.1 Spread-Spectrum

Spread spectrum RF transmission allows multiple transmitters to operate at the same frequency without interference. There is a practical limit the the number of transmitters which can be simultaneously transmitting but spread spectrum modulation is a useful method in light if the current US FCC regulations which allow low power transmitters in the 900 mHz frequency region.

Spread spectrum can be used for point to point or point to multipoint transmission. It is primarly used for point to point transmission.

Digital spread spectrum transmission communications systmes are an excellent candidate for use with the cdqPRIMA

1.2.3.2 Satellite Links

Satellite transmission is used for primarly for point to multipoint transmission. Such systems are used to broadcast to many listeners.

The cdqPRIMA is perfectly suited to work in digital satellite systems.

1.3 Compression Algorithms

1.3.1 CCS MUSICAM Digital Audio Compression

1.3.1.1 Introduction

Developments in the fields of consumer audio electronics and professional audio processing have been increasingly influenced by digital technology. Until five years ago, developments in the field of source coding were mainly restricted to the bit-reducing coding of speech signals for telecommunications applications.

Today, source coding techniques are playing an even greater role in the field of high quality digital audio. The reasons for this are the direct relationship between the low bit

rates associated with compression and the costs associated with the transmission and storage of compressed audio.

The bit-rate for high-quality stereo audio signals (1,411 kbs for a CD) can now be reduced by the MUSICAM algorithm to about 200 kbs. This is the result of major progress in the development of source coding techniques that utilize knowledge of the human ear. This means that the average quantization of the audio signal at a sampling rate of 44.1 kHz would be approximately 2 bits per sample in the mono channel instead of the 16 bits per sample used in CD's. Despite this high reduction in the bit rate, no quality differences are discernible to a trained ear. A slight impairment only becomes audible at higher compression rates. Additionally, MUSICAM offers the flexibility of independently adjustable audio sampling rates (32 kHz, 44.1 kHz, 48 kHz...) and digital bit rates (56 kbs, 64 kbs, 112 kbs, 128 kbs, 192 kbs, 256 kbs, 384 kbs...) as well as embedded data within the audio bit stream. All of these features are incorporated in the recently approved ISO MPEG audio standard. No other audio compression algorithm has undergone the scrutiny and testing subjected to MUSICAM as a result of the ISO selection process. The ISO standards committee has selected a truly universal digital audio source coding system with the flexibility to meet different system demands. Current and future audio systems adhering to the ISO MPEG audio standard will be able to interoperate easily and reliably. This will allow manufacturers to build sophisticated audio equipment and consumers to purchase hardware without the fear of obsolescence.

1.3.1.2 MUSICAM Compression Concepts

The main principle of MUSICAM is the reduction of redundancy and irrelevance in the audio signal. Every audio signal contains irrelevant signal components that have nothing to do with the identification of the audio signal (i.e., determination of timbre and localization). These irrelevant signals are not significant to the human ear and are not required by the information processing centers in the brain. The reduction of irrelevance means that these signal components are not transmitted. This results in a lower bit rate without any perceived degradation of the audio signal. Furthermore, it is possible to allow a certain degree of quantizing noise that is inaudible to the human ear due to the masking effects of the audio itself.Every audio signal produces a masking threshold in the ear depending on a time varying function of the signal. To understand this masking effect, the concept a masking tone must be defined. A masking tone is simply a high amplitude audio signal occurring over a relatively narrow frequency span and is often called a masker. Typically, in an audio signal there exists a number of these masking tones occurring at several different frequencies.

A masking tone renders smaller amplitude tones close to it inaudible due to its masking effect. The exact shape of the masking effect is called the masking threshold. The aggregate of all the maskers defines a global masking threshold and the parts of an audio signal below the global masking threshold are inaudible. They are said to be masked and therefore need not be transmitted. Other signal components above the masking threshold only require the level of quantization to keep quantization noise below the masking threshold, and thus the quantization induced noise remains inaudible. Quantization noise

can be better adapted to the masking threshold of the human ear by splitting the frequency spectrum into sub-bands.

The quantization of the analog time samples required for each sub-band is dependent on the minimum masking value in each sub-band. This minimum masking level is a measure of the allowed quantization noise that is just below the level of perceptibility. Sub-bands whose desired signals are well below the masking threshold (and are thus irrelevant for the human ear) do not need to be transmitted.

In each 24 millisecond period, a calculation of the masking threshold is performed for each sub-band. This threshold is then used to compute the psycho acoustically best allocation of the available bits. This process is called dynamic bit allocation. Audio data is quantized using the dynamic bit allocation and thus the required bit rate for time-variant audio signal's changes continuously due to the changing masking threshold. If there is an insufficient number of bits to hide the quantizing induced noise completely, then the noise is placed in the least objectionable place in the audio sample. If there is an excess number of bits, then the extra bits are used to reduce the quantizing induced noise to as low as possible level. The allocation of the extra bits is crucial and allows multiple encode-decode cycles as well as post production of the audio.

The total transmitted bit stream contains quantized audio values as well as auxiliary information describing bit allocation and scale factors, all of which are required by the decoder to reproduce the audio information.

The scale factors are determined by searching for the maximum sampling value in each sub-band and quantizing the result using 6-bit sampling. The scale factors have a dynamic range of 120 dB that is sufficient for future encoding for quantized PCM signals using up too 20-bit sampling yet still retain their dynamic range. All necessary information is encoded into MUSICAM frames each of which represents about 24 milliseconds of real-time audio.

All the complex calculations of the MUSICAM algorithm are performed by the encoder. Decoders are designed to be universal. MUSICAM decoders can be constructed which correctly decode and play back audio information that has been encoded by a range of MUSICAM encoders. This aspect of the MUSICAM algorithm is crucial because it enablesrefinements in the encoding process to further improve performancewithout impacting decoders that are already installed.

1.3.1.3 Performance Considerations

1.3.1.3.1 Introduction

Before discussing the various quality aspects of MUSICAM, it is necessary to define the terms used to represent the field of use of the audio. The 4 commonly discussed fields of use are:

- Contribution

- Distribution
- Emission
- Commentary

The term contribution grade is used to describe quality suitable for digital mastering. Its use would be in the transmission of a digital master from one archive to another. It is assumed that the original copy is in a 16 bit linear PCM format and it is to be compressed, transmitted, decompressed and stored in a 16 bit linear PCM format at the distant end. Because the audio is the source of future compression/decompression cycles, any contribution grade compression system must be able to withstand many encode-decode cycles and post production without any apparent degradation.

Distribution grade systems are used to transmit audio between two storage devices. However, the number of encode-decode cycles is limited to only a few.. Distribution grade systems are used when the number of audio compression-decompression cycles is limited.

Emission grade systems are used when there is only one compression-expansion cycle anticipated. This is the case when audio is compressed and transmitted from one place to another, decompressed and stored on an analog tape and the only future manipulations done are in the analog domain.

Commentary grade systems are used for transmitting voice grade audio.

These definitions make no mention of the analog bandwidth or the exact definition. They are vague terms used to describe ability of the audio to withstand multiple encode-decode cycles. In all cases, the compressed audio is assumed to be indistinguishable from the original.

1.3.1.3.2 ISO Background

The only independent measurements of audio quality of MUSICAM types of compression systems have been done by the MPEG ISO committee. Four algorithms in July of 1990 were tested and the winner according to the rules of the tests was MUSICAM. This algorithm was adopted and it was agreed that, to the extent possible, the best features of the second place algorithm, ASPEC, would be incorporated into MUSICAM to produce the final ISO standard.

The ISO committee decided to have a layered standard with 3 layers. Layer 1 is a very simplified version of the original MUSICAM algorithm. Layer 2 is essentially the MUSICAM algorithm as tested, and Layer 3 is a modification of Layer 2 that includes various features of ASPEC. It was anticipated that the resulting audio quality would improve with higher layer number. After the layers were defined, they were implemented according to the standard and each layer was tested in the May 1991 tests.

The results of these tests were surprising because Layer 3 scored lower than Layer 2. It has recently been decided that additional work on Layer 3 was needed and that layer

would be retested in December of 1991. Layers 1 and 2 have been frozen in their present state because they have met their design objectives. As a result of the ISO effort, the MUSICAM algorithm is now properly called the MPEG Layer 2 compression algorithm.

It is clear from the most recent ISO tests that no compression scheme performs acceptably at 64 kbs. Work at that bit-rate is the subject of further research and will be addressed in a future standard.

The intensity or joint stereo mode of compression supported by Layer 2 (called Layer 2A) was not tested during the May 1991 tests.It is important to recognize that the ISO tests have provided a wealth of knowledge about the MPEG Layer 2 algorithm. Other algorithms such as SEDAT, AC-2 and APT-X did not even participate in the ISO tests and their strengths and weaknesses are unknown. It is certainly clear that MPEG Layer 2 has been demonstrated to be a superior algorithm. This claim can be supported by a large body of test data. Other algorithms have little or no independent test data to substantiate their quality claims.

1.3.1.3.3 Quality vs. Bit Rate

The MUSICAM design allows the digital bit-rate, analog bandwidth and quality to be generally related by the formula

$$\text{Digital Bit-Rate Quality} = \frac{\text{Digital Bit-Rate}}{\text{Analog Bandwidth}}$$

As indicated above, the quality increases as the bit-rate increases and the analog bandwidth is kept constant. Similarly, if the digital bit-rate is kept constant, and the analog bandwidth is decreased, then the quality improves.

The ISO test in Stockholm in May 1991 has demonstrated that at a digital bit rate of 256 kbs per stereo channel; MPEG Layer 2 is statistically identical to the original signal. This means that the panel of approximately 60 highly trained listeners could not distinguish the original uncompressed source material from the audio compressed by the MPEG Layer 2 algorithm. The conclusion of the ISO tests (at 256 kbs per stereo channel) was that MPEG Layer 2 is transparent. MPEG Layer 2 scored 5 on the MOS (mean opinion score) scale where the lowest is 1 and the highest score is 5.

It is important to note that no other algorithm tested at ISO (including ASPEC) was considered transparent in the 256 kbs stereo tests. The ISO tests were conducted on stereo channels composed of two mono channels so that the combined bit rate was 256 kbs per stereo channel. The audio quality at 192 kbs was determined by ISO to be 4.5 on the MOS scale using stereo encoding and 2.0 for a mono channel at 64 kbs.

The MPEG Layer 2 algorithm provides the following qualities at various bit rates.

- contribution 384 kbs (stereo, Layer 2)
- distribution 256 kbs (stereo, Layer 2)

- emission 192 kbs (stereo, Layer 2A)
- commentary 64 kbs (mono, Layer 2)

The classification of 192 kbs for the emission grade is based on recent work at the IRT (Institute fur RundfunkTechnique) and relies on the intensity (joint) stereo coding technique for additional compression.

1.3.1.4 Tolerance to Transmission Errors

The ISO MPEG Layer 2 data block consists of two parts. The first is the header and consists of framing, bit allocation, scale factors and other side information. The second part of the frame is the audio data. In the case of 256 kbs per stereo channel, the length of a 24 millisecond frame is 6144 bits, the header part of the frame is approximately 300 bits and the remainder of the frame is the audio data. The bit integrity of the entire header is vital since it defines the layout of the remainder of the frame. Any bit error in the header causes degradation because the following parts of the frame would be decoded incorrectly and thus 24 milliseconds of audio would be lost.

An error in the data part of the frame can range from imperceptible to just barely noticeable. This is because a single bit error only affects a single data sample and thus only a very small time. If the bit error occurs in the least significant bit of the data sample, the effect of the error is minimal. However, if the error occurs in the most significant bit (the sign bit) then the effect is more pronounced.

The header of an MPEG frame is protected by an error protection polynomial and provides the ability to detect errors that occur in the header. The data part of the frame is unprotected and any error occurring in the data part of the frame remains. The error strategy used for the ISO MPEG system is as follows. If an error is detected in the header, the last frame (24 milliseconds) of audio is repeated. If, in the succeeding frame, an error is detected in the header, the second and all succeeding frames with errors are muted. This error mitigation technique has been shown to be effective for bit rates of approximately 10-5. This error rate represents error rates easily achievable by transmission systems. Using this strategy, there is a smooth degradation of the audio quality as the error rate increases until the error rate becomes excessive at this point the audio output mutes.

1.3.1.5 Tolerance to Multiple Processing

To understand the effect of multiple encode and decode cycles it is important to review the predominant effect that allows MPEG audio to achieve its compression. This is the hiding of quantization noise under a loud signal. MPEG audio adjusts the degree of quantization induced noise in each sub-band and thus hides more noise (uses fewer bits) in the sub-bands that contain large amounts of audio energy.

The quantizing noise raises with each encode and decode cycle and after a sufficient number of cycles, the noise level becomes perceptible. The degradation process is gradual

and depends upon level of the quantizing noise on the original. For example the following table list the approximate numbers of total encode and decode cycles before the noise

Bit Rate	*Number of*
384 kbs	15
256 kbs	5
192 kbs	2
128 kbs	1

Table 1-1

Number of transcodings vs bit rate

becomes significant.

It is important to understand that these are approximate and the exact number depends highly on the source material.

1.3.1.6 Post Production Processing Effects

Post production processing of compressed audio is a complicated effect to model. For example, an equalizer changes the level of a range of frequencies, while limiting and compression are non-linear processes. Very little test data is available to ascertain the effects of post processing. Private communications with the IRT suggest that MPEG layer 2 is robust against the effects of post processing and the degree of robustness depends on the compression rate. In particular, 384 kbs audio is unaffected by post processing while 128 kbs audio is somewhat sensitive to post processing. It is not easy to define tests to measure the effects of post processing but an international standards body (CCIR) is specifically designing test to determine the effects of both transcoding and post processing. These tests were conducted in November of 1991 and represented the first time such tests were performed by an independent organization.

MPEG audio represents the most tested, documented and reviewed audio compression algorithm in the world. It is significant to note that no other compression technique has survived this crucial review process as well as the MPEG algorithm and, many other algorithms have elected not to participate in this review process. It is precisely these untested algorithms that make the boldest claims. MPEG audio provides the security of the international review process to insure the highest quality audio possible with today's technology.

1.3.1.7 The MUSICAM Advantage

The MUSICAM digital audio compression algorithm has been designed to take advantage of future advances in psycho acoustic research. To make this possible, the decoder is designed to be a slave to the encoder. This technique allows the entire system to be upgraded by simply changing the encoder software. Once this change is made, the entire network is upgraded and the encoder enhancements are reflected at the output of all decoders.

The MUSICAM algorithm is designed to operate at multiple bit-rates. This gives the user the ultimate flexibility to make the tradeoff between quality and cost. The use of higher bit-rates (384 kbs) allows nearly an arbitrary number of transcodings and extensive post processing while still maintaining transparency. The middle bit-rates (256-192 kbs) allow lesser amounts of manipulation while the lower bit's rates (128 kbs) are the most sensitive the these effects. As advances in the research progress, today's bit-rates required to achieve a desired quality will decrease and the ease of MUSICAM to accommodate these advances provides a significant advantage. This is being demonstrated by the research into intensity coding of stereo signals. This shows that the data rate of 192 kbs for stereo signals will most likely be the new standard rate for transparent audio and will supplant the 256 kbs rate accepted as the standard today.

MUSICAM is able to embed other information within the audio bit stream. Again, in the MUSICAM design, the data rate of this ancillary information is completely flexible and thus is entirely in the hands of the system designer. This data rate is completely determined by the encoder and thus the may be changed at any time with no modifications to the decoders. The inclusion of data in the audio bit stream reduces the bits available for audio data and thus the system designer can make the delicate tradeoff between the ancillary data rate and audio quality.

The flexibility of MUSICAM to adapt to current and future needs is a powerful feature necessary to prevent the obsolescence of any system based on it. There is now no need to divine future system needs because the system can be easily be changed to accommodate its ever changing requirements.

Ancillary Data Port

The CDQPRIMA provides for transmission of asynchronous data via a RS-232 interface. This interface provides a transparent channel for the transmission of 8 data bits. The data format is 1 start bit, 8 data bits, 1 stop bit and no parity bits. This interface is capable of transmitting at the maximum data rate selected by the encoder and decoder data rate dip switches and thus no data pacing such as XON/XOFF or CTS/RTS is provided. Appendix C describes the encoder and decoder dip switches.

The encoder RS-232 data rate can be set from 300 to 19,200 bps. The use of the ancillary data channel decreases the number of bits available to the audio channel. The reduction of the audio bits only occurs if ancillary data is actually present. The data rate can be thought of as a maximum data rate and if there is no ancillary data present, then no data bits are transmitted. A typical example of this situation occurs when the CDQPRIMA

encoder is connected to a terminal; when the user types a character the character is sent to the decoder at the bit rate specifiea.

The setting of the decoder baud rate sei:ction dip switches must be done considering the setting of the encoder. The decoder dip switches must be an equal or higher baud rate relative to the encoder. For example, it is possible to set the decoder ancillary baud rate to 9,600 baud. In this case, the encoder baud rate may be set to any value from 300 to 9,600 but not 19,200. If the decoder baud rate is set to a higher rate than the encoder, the data will burst out at the decoder's baud rate. The maximum sustained baud rate is controlled by the encoder.

The algorithm for the transmission of ancillary data is for the encoder to look during each 24 millisecond MUSICAM frame interval and see if any ancillary data is in its input buffer. If there are characters in the encoder's input buffer, then the maximum number of characters consistent with the selected baud rate are sent. During a 24 millisecond period,

Bit Rate	*Number of Characters*
300	1
1200	3
2400	6
3600	9
4800	12
7200	18
9600	24
19200	47

Table 1-2

Number of characters/frame (48 kHz)

the table below shows the maximum number of characters sent for each baud rate.

The CDQPRIMA provides no error detection or correction for the ancillary data. The user assumes the responsibility for the error control strategy of this data. For example, at an error rate of 10-5 (which is relatively high) and an ancillary data rate of 1200 baud, 1 out of every 83 characters will be received in error. Standard computer data communication protocol techniques can be used to maintain data integrity.

When designing an error protection strategy, it must be remembered that the CDQPRIMA may occasionally repeat the last 24 milliseconds of audio under certain error conditions. The effect on the audio is nearly imperceptible. However, the ancillary data is not repeated.

1.3.1.8 Compatibility with older CCS CODECs

1.3.1.8.1 CCS Old

See dave brown for an explaination.

1.3.1.8.2 CCS New

See dave brown for an explaination.

1.3.2 Layer 3

ISO MPEG Layer 3 was an attempt of the ISO committee to utilize the best featurs of the algorithm which lost the ISO competition (ASPECT) with the winning algorithm (MUSICAM). The resulting algorithm utilizes the sub-band filter bank of MUSICAM with MDCT within each sub-band. The results of ISO and CCIR testing have shown that Layer 3 provides a small advantage only at 64 kbs mono and has the distinct disadvantate when cascaded. It is an extreemely complicated algorithm and provides limited improvement at best.

1.3.3

The CdqPRIMA uses Adaptive Differential Pulse Code Modulation (ADPCM) to reduce the digital bit rate needed to transmit the digital representation of an analog signal. The CdqPRIMA digitizes the incoming analog signal with a 16 bit linear Analog to Digital converter (AD) 16,000 times per second. The Nyquist theorem states that at this sampling rate, an analog signal of up to 8,000 Hertz can be reconstructed from the sampled signal. Using this sampling rate and AD converter resolution, the following uncompressed bit rate is derived:

PCM bit rate = 16,000 * 16

PCM bit rate = 256,000 bits per second

The cdqPRIMA then compresses this bit rate down to 64,000 or 56,000 bits per second using ADPCM.

To accomplish this compression, ADPCM utilizes the fact that the next sample of speech can be predicted by previous speech samples. The CdqPRIMA only transmits the difference between the predicted and actual sample. If the prediction process is effective, then the information to transmit consists of significantly fewer bits than the digital representation of the actual sample. The prediction accuracy is greatly enhanced by splitting the 8 kHz band into two 4 kHz bands. The signal in each band is predicted separately. This allows a more faithful representation of the analog signal then is possible by considering the whole 8 kHz band at once.

In conventional PCM, the binary representation of each sampled analog point is used. Differential PCM (DPCM) transmits the difference between the previous point and the

current point. In this scheme, the prediction process only involves the previous point. In fact, the predicted value of the current point is exactly the last point. In CCITT G.722 implementation of ADPCM, the predictor is very sophisticated and uses the previous 6 points to predict the current point. This results in a very accurate prediction and hence a very low bit rate.

1.3.4 Future Algorithms & Prima Upgrade Capacity

The cdqPRIMA has the capability to hold several audio compression algorithms. This permits the cdqPRIMA to be resistant to obselence. The cdqPRIMA can be downloaded from ISDN and thus the future upgrades are simple and effortless to install. This should be contrasted to the ROM type of update procedure currently employed by most CODEC manufactures.

2. Installation

2.1 Unpacking & Inspection

Upon opening the shipping container, examine the cdqPRIMA for mechanical defects. Report any problems promptly to CCS. Plug the unit into the main power and turn on the unit via the rear panel power switch. The front panel LCD's should illiminate and display the power up sequence on the front panel LCD display.

2.2 Location of Units

The cdqPRIMA has been designed to allow installation at locations with high RF fields.

2.2.1 Environmental Considerations

It is important that the ambient temperature specifications are met. It is usually possible to stack the cdqPRIMA units directly on top of other electronic equipment. It is important that the cdqPRIMA not be exposed to condensing humidity or fungal environments..

2.2.2 Configuration Dependencies

The cdqPRIMA can be used with a variety of digital transmission facilities. Typical applications consist of ISDN, satellite and dedicated facilities. The cable lengths for the interconnections can be from centimeters to kilometers. It is important to utilize twisted pair cable with an overall shield for the compressed audio interface. Flat ribbon cable should be avoided!

The digital audio interconnections are much less tolerant to longer cable lengths. Distances of 30 meters should be considered as an upper bount. Good cable construction is a necessity for the digital audio cables.

2.2.3 Remote Control Considerations

The cdqPRIMA is designed to be completely controlled remotely by a host computer. A rich command set can be used to control the entire operation of the cdqPRIMA. The section entitled cdqPRIMA Remote Control Commands contains a detailed description of all the remote control commands.

2.3 Connection to Network

The cdqPRIMA family provides a variety of digital interfaces. Including V.35, X.21 leased circuit and RS422. Each of these digital interfaces requires clock and data to be exchanged between the cdqPRIMA and the terminal equipment. The cdqPRIMA always expects the clock to be provided by the terminal equipment. The encoder section outputs data synchronized with the clock and the decoder expects the data to be synchronized

with the clock. Figures 5 and 6 show the interconnection of the cdqPRIMA to a generic piece of terminal equipment. The timing relationships are shown in Appendix B.

The data and clock lines are differential requiring a pair of wires for each signal. The control lines in the V.35 interface are single ended and require only one wire for each signal. The X.21 control lines are differential. The RS422 interface does not support any control lines. Any input control lines defined are ignored by the cdqPRIMA and any output control lines defined are held at constant values. See Appendix A for the

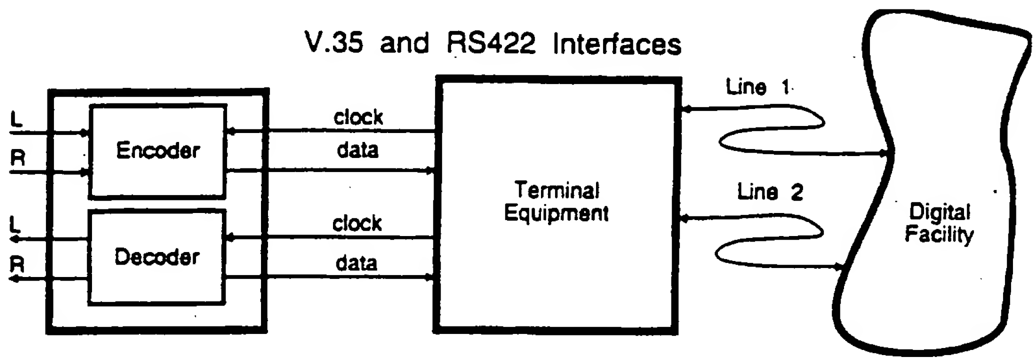


Figure 5
Basic interconnection to digital network

Figure 2-1

Basic interconnection to digital network - RS422/V.35

definition the pins used for each type of interface.

Each interface defines a voltage level for each of the signals. In the case of V.35 and X.21, a connector type is also defined. The connector defined in the V.35 specification is not used by the cdqPRIMA because of its size. Instead, a smaller DB25 connector is used. In the case of the V.35 interface, the cdqPRIMA conforms to the electrical specification but requires an adapter cable to convert the DB25 connector to the connector specified in the V.35 specification. The connector and the pin-out chosen for the V.35 interface in the CDQPRIMA are a common deviant found in many systems. It is important to remember that V.35 has a separate clock for transmitted and received data. Appendix E describes the pin-out required for a DB25 to V.35 connector. The RS422 interface specification only defines the electrical voltages at the interface and leaves the pin-out and meaning of the pins to the hardware designer. The RS449 interface specification utilizes the electrical specifications of RS422 but specifies a mechanical connector. RS449 also specifies numerous control signals besides clock and data. The cdqPRIMA RS422 interface pin-out is specified in Appendix A. The RS422 interface also has a separate clock for the transmitted and received data. The cdqPRIMA RS422

interface also echoes the transmitter clock. If the terminal equipment clocks the encoder data with the echoed clock, then the cdqPRIMA may be located up to 4000 feet from the terminal equipment without having to worry about the encoder to clock skew.

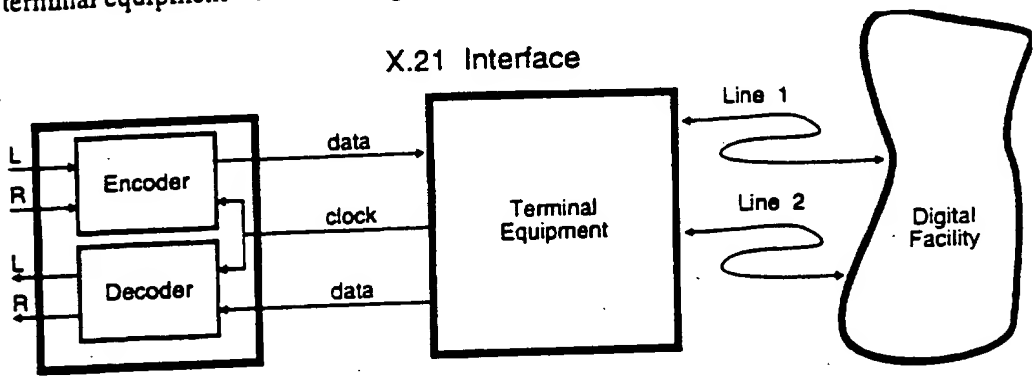


Figure 6
Basic interconnection to digital network

Figure 2-2

Basic interconnection to digital network - X.21

The X.21 interface specification is in general a very complex specification. The general specification allows a mechanism for communication between the customer equipment and the network. This communication path can be used for things such as dialing. A sub-set of the specification, called the leased circuit, restricts the interconnection to only clock and data and a very simple control signal. The mechanical connector required is the DB15 with the pin-out specified in Appendix A. The electrical specification is RS422. The X.21 interface has only one clock for both the transmit and received signals.

Since the X.21 utilizes the RS422 electrical interface, the cdqPRIMA can use the same connector for both interfaces. In the case of the X.21 interface, the single clock is used internally for both the transmit and received timing. The selection of the type of digital interface is governed by rear panel dip switches. See Appendices C and D for the appropriate settings.

2.3.1 ISDN Card

The ISDN interfaces provide the following capabilities

- TA101 1 BRI S/T interface
- TA201 1 BRI S/T interface
- TA202 2 BRI S/T interface

- TA211 1 BRI U interface (US only)
- TA222 2 BRI U interface (US only)

- The TA101 provides basic ISDN TA functions and requires a separate ROM for each contry

2.3.2 Interface

The V.35 interrface used on the cdqPRIMA utilizes the standard voltage levels and signals of the V.35 standard. It utilizes DB15 connectors instead of the standard large multipen connector. A cable adaptor is available which adapts the DB15 to the standard connector.

2.3.3 Interface

The X.21 interface provides the voltage levels, pinout and connector specified in the X.21 specification.

2.3.4 RS422 Interface

The RS422 interface utilizes the same DB15 connectors and voltage levels as used in the X.21 interface. It replaces the X.21 Control and Indicator signals with other timing signals.

2.4 Rear Panel Connectors

2.4.1 Analog I/O

The cdqPRIMA provides 18 bit AD and DA converters for the analog conversion modules. The analog sections of the cdqPRIMA are set to +18 dBu maximum input levels. Other analog input and output levels are possible by consulting CCS..

2.4.2 AES/EBU I/O

The AES/EBU digital audio interface standard provides a method to directly input (and output) audio information. This standard allows interconnection of equipment without the need for Analog/Digital conversions. It as always desirable to reduce the number of AD conversions since each time the conversion is performed, noise is generated. The cdqPRIMA allows digital audio input and output via a rear panel connector.

The cdqPRIMA model 1xx series, the AES/EBU connector is a DB9 due to space considerations. The cable drawing for an adaptor from the DB9 to standard XLR connectors is provided in the section labeled CABLE DRAWINGS.

The cdqPRIMA 2xx series uses the standard XLR connectors.

The AES/EBU digital input is rate adapted on onput as well as output to eliminate any digital clock problems. The AES/EBU digital output from the decoder can be synchronized to a studio clock via an external AES/EBU sync input located in the rear of the cdqPRIMA

Because of the rate adaptors, the input/output digtial rates are not required to be the same as the internal rates. For example, it is possible to input 44.1 kHz AES/EBU digital audio input and ask the cdqPRIMA to perform compression at 48, 44.1 or 32 kHz (by using the front panel LCD display or the remote control **ESR** command). This is possible because the digital audio rate adapters.

Digital audio input sources can only be 32, 44.1 or 48 kHz. These input sampling rates are automatically sensed and rate adapted.

The compression algorithm at the encoder determines the digital sampling rate at the decoder. Thus the **ESR** command sets the internal sampling rate at the decoder. The AES/EBU digital output signal at the decoder is determined by the **DDO** command and can be a varity of values. See the **DDO** command for a detailed description.

The encoder receives direct digital input via the connector on the rear panel. Analog or digital (but not both simultaneously) signals may be input to the cdqPRIMA as selected by the front panel switch. If the digital input is selected, the CDQPRIMA locks to the incoming AES/EBU input and displays the lock condition via a front panel LED (not available on all models). If digital audio input is selected, the AES PLL lock light must be illuminated before audio is accepted for encoding. In normal operation, the CDQPRIMA locks its internal clocks to the clock of the telephone network. For loopback, it locks its clocks to an internal clock. In either case, the clock used by the CDQPRIMA is not at precisely the same frequency as the AES/EBU input. To prevent slips from occurring due the presence of two master clocks, a rate synchronizer is built into the encoder section to perform the necessary rate conversion between the two clocks.

The decoder outputs direct digital signals via the rear panel connector. Additionally, the decoder may be synchronized to an external clock by an additional connector (SYNC) on the rear panel. If no input is present on the decoder AES/EBU SYNC input line, then the output AES/EBU digital audio is generated by the internal clock source that is either at the telephone or internal clock rate. If the SYNC input is present, then the digital audio output is generated at the frequency of the SYNC input. The presence of a valid sync source is indicated by the illumination of the front panel AES PLL LED. The sync frequency may be slightly different from that of the CDQPRIMA clock source and again rate synchronism is performed to prevent any undesired slips in the digital audio output. The SYNC input is assumed to be an AES/EBU signal with or without data present. The CDQPRIMA only uses the framing for the frequency and sync determination.

2.4.3 Power & Power Switch

This switch is used to control the main power to the cdqPRIMA.

2.4.4 Remote Control

This I/O port on the cdqPRIMA provides for either RS232 or RS485 remote control. It has the same capabilities as the front panel remote control. The choice of the RS232 or RS485 interface can be made by a remote control command or a front panel LCD command. A detailed description of the remote control commands is given in section entitled A Summry of cdqPRIMA Remote Control Commands.

2.4.5 Ancillary Data

The Ancillary Data connector provides an RS232 bi-directional interface for the transmission of asynchronous data. The data rates range from 300 to 38400 baud.

2.4.6 Alarm

This is a DPDT relay output whose function is controlled by the RLS action. See the section entitled cdqPRIMA Logic Language. It is often used as a summary alarm output to indicate the failure any major subsystem in the cdqPRIMA.

2.4.7 1xx Series

2.4.7.1 Opto/Relay I/O and Sync Data

For space reasons, the 4 optical isolated inputs, 4 relay outputs and the synchronous ancillary data I/O has been combined into one connector.

2.4.8 2xx Series

2.4.8.1 Optical)

The cdqPRIMA (on the 2xx models) provides an optional optical digital audio interface This interface utilizes the EIA-J optical connectors. The functions of the EIA-J optical inputs are identical to the AES/EBU digital input connectors described above. The EIA-J connectors are enabled by a a rear panel slide switch.

2.4.8.2 Time Code

The cdqPRIMA allows the transmission of timecode at rates of 24, 25, 29 and 30 frames per second. The cdqPRIMA automatically detects the presence of timecode at the encoder, converts it into a digital form and then multiplexes it into the ancillary data stream for transmission with the audio. At the decoder side, the ancillary data is seperated from the audio and then demultiplexed. The time code is reconstructed

2.4.8.3 Opto Inputs

The optically isolated inputs on the 2xx series are identical to that of the 1xx series except that there 8 input sources.

2.4.8.4 Relay Outputs

The relay outputs on the 2xx series are identical to that of the 1xx series except that there 8 input relays.

2.4.8.5 Sync Data

The synchronous data port on the 2xx series is similar to sync data port on the 1xx series except that the output can be RS232 as well as RS484.

2.4.8.6 RS232

The RS232 I/O connector is used to provides an additional port into the data multiplexor. It can be thought of as a second RS232 ancillary data port.

2.4.8.7 RS485

The RS485 I/O connector is used to provides an additional port into the data multiplexor. It is a dedicated RS485 port and can be used to control RS485 equipment.

3. Feature Summary

3.1 Async Ancillary data

Associated Remote Control Commands

CAN Set ancillary data mode
CMA - Set MUX ancillary data baud rate
CDR Set ancillary data rate for encoder and decoder DSP
DSB Set decoder synchronous ancillary data bit rate
ESB Set encoder synchronous ancillary data bit rate

The ISO-MPEG audio packet consists of of the following parts:

- Header
- Audio Data
- Ancillary Data

If the sampling rate is 48 kHz, then the length of each packet is 24 milliseconds. The header consists of a 12 bit framing pattern, followed by various bits which indicate the data rate, sampling rate, emphasis, copyright, origional ... . These header bits are protected by an optional 16 bit CRC.

The Header is followed by the audio data which describes the compressed audio signal.

Any remaining bits in the packet are considered ancillary data. The format of the ancillary data is user defined. CCS has defined two ways of using the ancillary data. The first method has been used in the CDQ20xx series products and treats the entire data stream as one logical (and physical) stream of data.

The cdqPRIMA series supports the older CDQ20xx ancillary data format as well as the newer cdqPRIMA format. This newer format allows the multiplexing of various logical and diverse data streams into one physical data stream. For example, switch closure, RS232 and time-code data are all multiplexed into a single physical data stream and placed in the ancillary data stream of the ISO MPEG packet.

The data rate from the Ancillary Data Multiplexor to the Encoder (and from the Decoder to the Ancillary Data Demultiplexor) is set by the **CDR** command. The data rate from the Ancillary connector into the Ancillary Data Multiplexor (and from the Ancillary Data Demultiplexor) is set by the **CMA** command. If **CAN** mode 2 is in use, then the **CMA** command has no meaning since the ancillary data is routed directly to and from the DSP's.

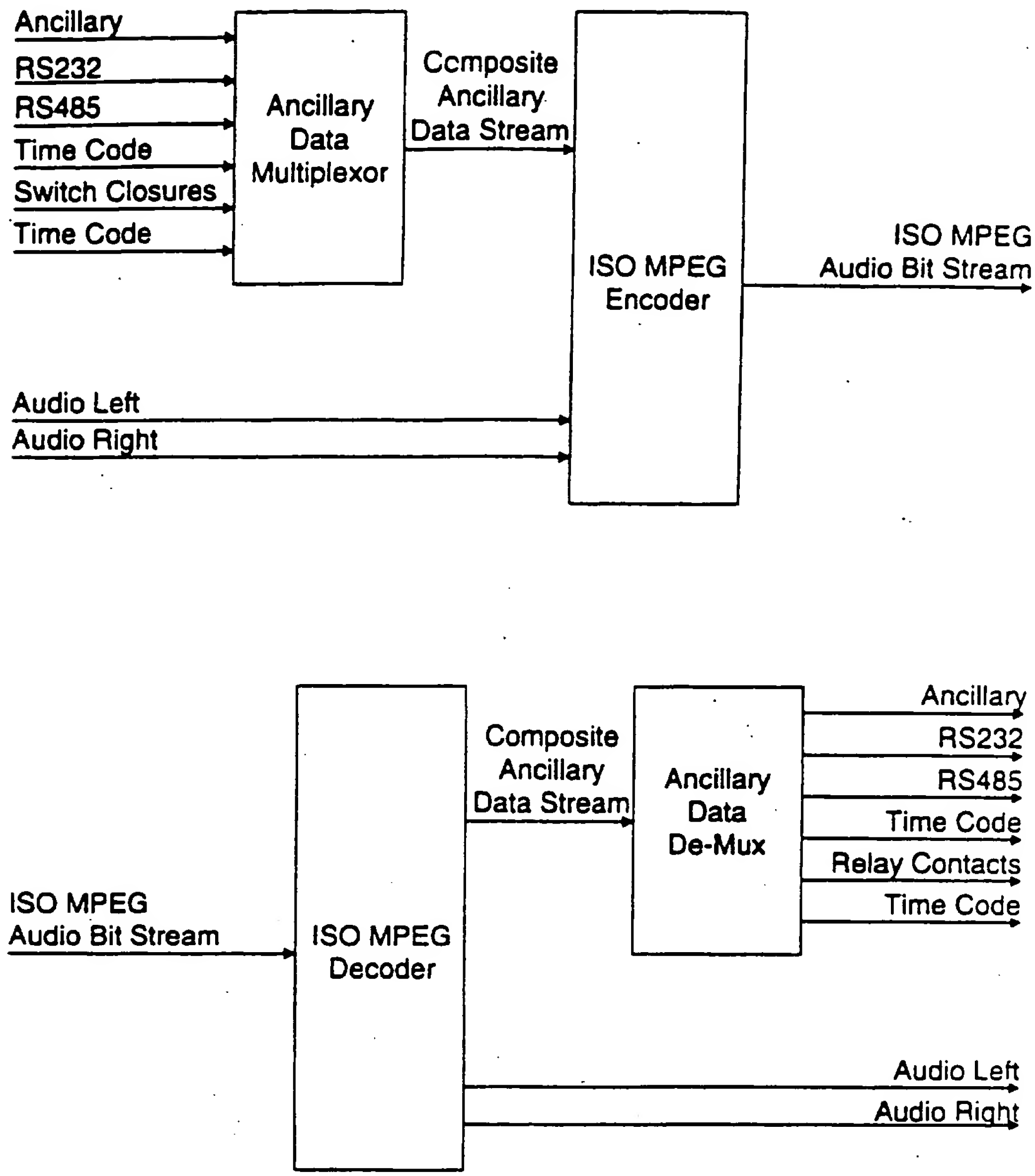


Figure 0-1

cdqPRIMA ancillary data overview

The synchrouous ancillary data rates are controlled by the **DSB** and **ESB** commands.

The communication between the MUX and the encoder DSP and the DE-MUX and the decoder DSP is via an asynchronous communications channel. The data rate of both of these channels are simultaneously set by the **CDR** command

The RS232 ancillary data port can be used in several ways. It can be connected through the MUX/DE-MUX as described above or it can be connected directly to the encoder and decoder DSP's. Connecting directly to the encoder and/or decoder DSP's allows the highest baud rate (38,400) to be used but remove many useful features of the MUX. The output of the MUX may be connected directly to the DE-MUX and bypass the encoder and decoder DSP. This configuration is useful for testing.

mode	*description*
0	Direct connect - encoder DSP only
1	Direct connect - decoder DSP only
2	Normal mux mode
3	Direct connect - input to encoder DSP and output to decoder DSP (old CDQ20xx mode)
4	Input to MUX and direct output from decoder DSP
5	Direct input to encoder DSP and decoder DSP output to DE-MUX
6	Normal mux mode - DSP bypass

Table 0-1

Summary of **CAN** modes

Mode 2 is the normal mode of operation when the data multiplexor is desired. Mode 3 bypasses the data multiplexor and connects the data at the Ancillary connector directly to the encoder and decoder DSP's. Mode 6 is useful for testing since it connects the multiplexor directly to the demultiplexor and thus bypasses the encoder and decoder DSP's.

3.1.1 Asynchronous ancillary data configurations

These various configurations are shown below.

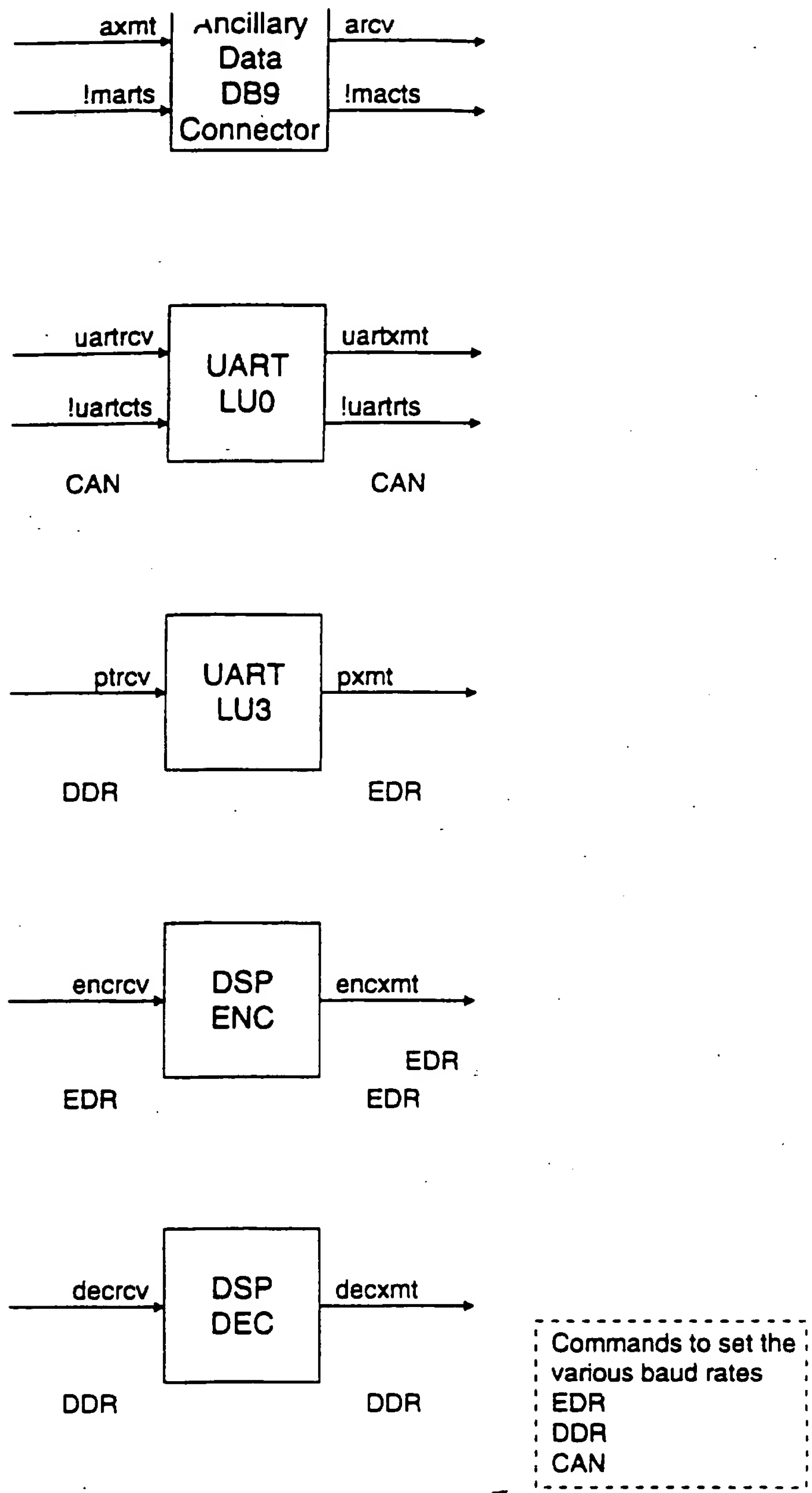

axmt
!marts
Ancillary Data DB9 Connector
arcv
!macts
uartrcv
!uartcts
UART LU0
uartxmt
!uartrts
CAN
CAN
ptrcv
UART LU3
pxmt
DDR
EDR
encrcv
DSP ENC
encxmt
EDR
EDR
EDR
decrcv
DSP DEC
decxmt
DDR
DDR
Commands to set the various baud rates
EDR
DDR
CAN

Figure 3-2

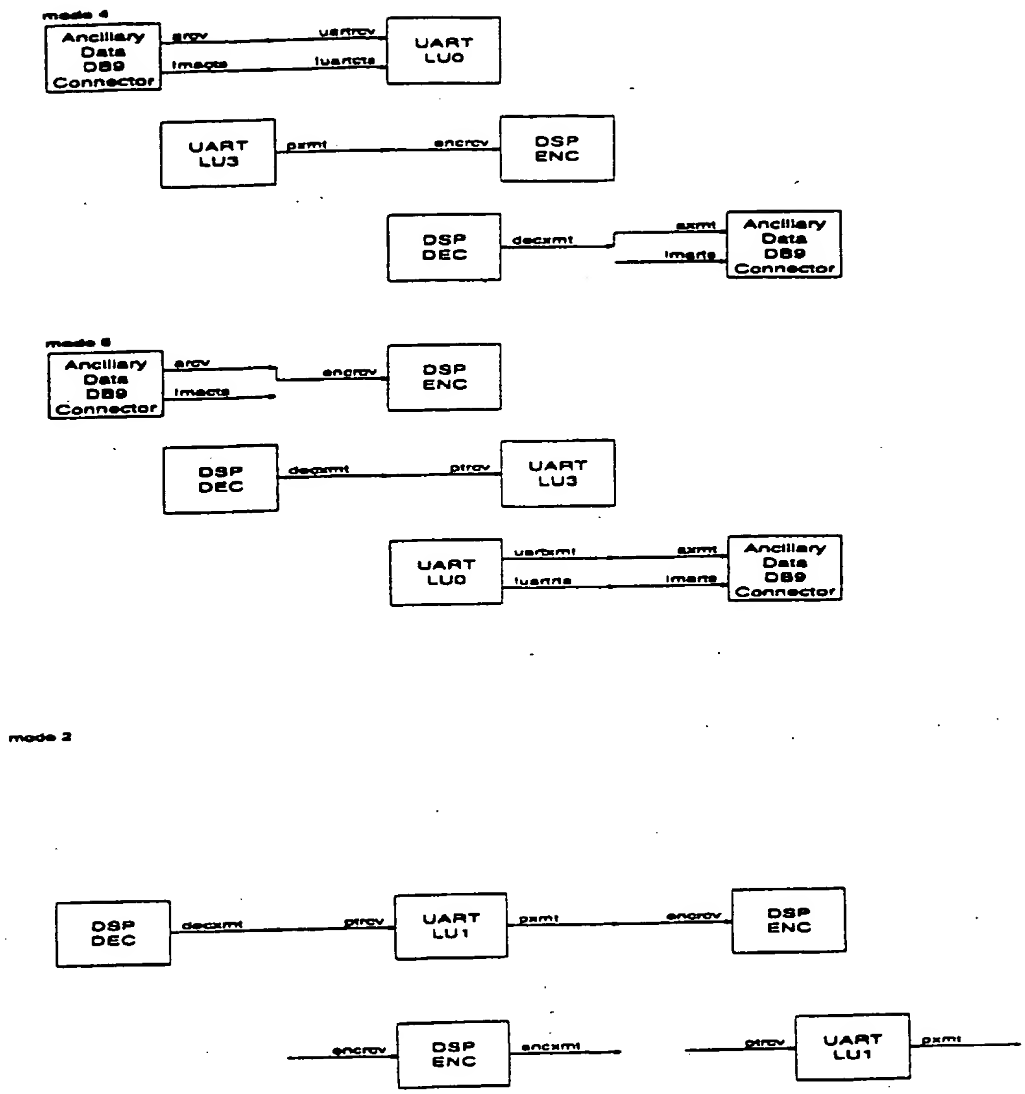


Figure 3-3
cdqPRIMA ancillary data switch configurations

mode 6

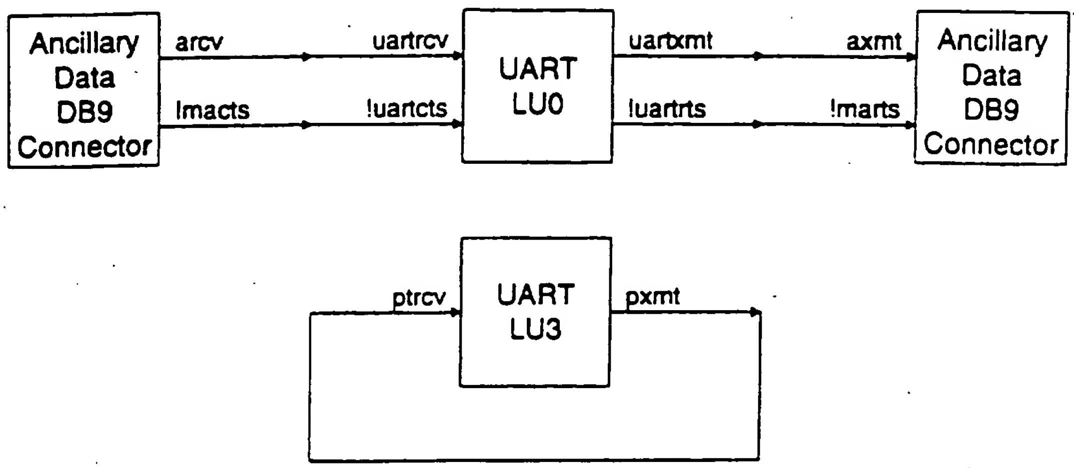


Figure 3-4

3.2 The Bit Error Rate Detector

Associated Remote Control Commands

MBC	Display BER counter
MBD	Set BER down count rate
MBL	Set BER count rate limit
MBR	Reset BER counter
MBU	Set BER up count rate

The bit error rate detector provides a method of monitoring the number of bit errors in the digital transmission path. The bit error rate detector is used when ISO frame protection is enabled. When each ISO/MPEG frame is received (every 24 milliseconds for 48 kHz sampling), the header CRC is checked for validity. If the frame header has a valid CRC. then the BER counter is incremented by a BER up count (any number from 0 to 9). If the frame is invalid, then the BER counter is decremented by the BER down count (any number from 0 to 9).

If the BER up count is set to 1 (by the **MBU** command) and the down count is set to 0 (by the **MBD** command), then the BER counter counts the total number of frames in error. If the up counter is set to 2 and the down count is set to 1, then the BER counter is sensitive to burst errors but not random errors.

The BER counter is compared to the BER threshold (set by the **MBL** command) to see if the counter is above or below the threshold. Actions such as closing a relay, dialing a phonenumber or lighting a LED or displaying a scrolling message can be taken.

The BER counter can be reset to 0 by the **MBR** command.

The current contents of the BER counter can be displayed by the **MBC** command.

3.3 Decoder

Associated Remote Control Commands

DAL	Set decoder algorithm
DBR	Set decoder bit rate
DCO	Set decoder decoding mode
DCS	Set channel copy/swap mode
DDA	Calibrate the DA converter
DIN	Set decoder - encoder interaction
DLI	Set decoder digital line format
DMD	Set decoder maintenance diagnostic mode
DMU	Mute decoder output channels
DSP	Scale factor protection
DRS	Print real-time decoder status bits

The decoder may be operated independently from the encoder by the proper setting of the **DIN** command. This can be extreemly useful if the cdqPRIMA decoder is operated in a

stand alone mode and is not controlled by the encoder. This stand alone mode can be use at any time. There are certian times when the decoder must operate in conjunction with the encoder. For example, when J.52 line bonding is used.

The realtime decoder status bits for the ISO/MPEG algorithm are displayed by the **DRS** command. The status bits displayed are the ISO frame header bits which are set by the encoder (See Encoder Header).

The audio output of the decoder can be muted by the **DMU** command.

The decoder audio output is controlled by the **DCS** command. This allows the swapping fo the left and right channel audio output. It also allows the left channel to be copied to the right channel (left channel mono) or the right channel to be copied to the left channel (right channel mono).

The decoder Digital to Analog (DA) converter can be calibrated by the **DDA** command. This calibration process insures that the DA converter is operating properly.

The ISO/MPEG scale factors can be protected by a CRC. This feature is controlled by the **DSP** command. In general, it is better to use scale factor protection if the data channel is noisey (high BER). If scale factor protection is enabled in the decoder, it must also be enabled in the encoder (**ESP**) or else the decoder output will mute.

The decoder can be instructed to decode only ISO/MPEG layer 2 bit streams by the **DCO** command. This is useful for determining if the incomming bitstream is fully ISO/MPEG compliant.

The decoder provides a method of generating test tones. The frequency and level of these tones are controlled by the **DMD** command.

If the decoder is operated in the stand alone mode (by setting **DIN** to YES), then there are several commands which must be set to determine the operation of the decoder. The first of these is the decoder bit rate. This is the compressed data rate and is set by the **DBR** command.

The DLI command is used to set the line format and is set by the **DLI** command. The DLI command instructs the decoder how to interperate the incomming compressed digital data. For example, if the incomming data is only present on digital interface 1 (DIF 1) then **DLI** L1 instructs the decoder to receive the data on that line.

The decoder algorithm is another parameter which is meaningful only in the decoder independent mode. The **DAL** command sets the decoder algorithm. This forces the decoder to operate utilizing a particular decompression algorithm.

3.4 Digital Interface

Associated Remote Control Commands

CDT Set state of the DTR/CON line

CIF Set digital interface type

The compressed audio digital interface (DIF) type is defined by this command. The are several types of digital interfaces. The types are TA and non-TA types. A TA type of digital interface is one that is capable of connecting to the ISDN line and can dial. The states of a TA type interface are

- DISCONNECTED
- DIALING
- CONNECTED

For cdqPRIMA models with the LED display, the states of the digital interface is shown by the 6 DIF LED's. If the LED is dark, then the state of the DIF is disconnected. If it is blinking, then the DIF is dialing and if the LED is illuminated, then the DIF is connected.

A non-TA interface is always in the connected state and there are several types of these interfaces. A list of these interfaces is shown below.

- X.21
- RS422
- V.35

The RS422 and X.21 have the same voltage levels and thus are both on the same interface card. This distinction between them is made by setting jumpers on the card.

The V.35 standard specifies different voltage levels and hence must use different type of line interface IC's. The interface card used for this standard is different from the interface card for the RS422/X.21 standard.

The **CIF** command (and corresponding LCD command) is used to define the type of digital interface to be used.

On the non-TA interfaces, there is a signal designated DTR for the V.35 interface and CON for the X.21 interface. These are control lines from the cdqPRIMA interface card to the external terminal adaptor equipment. The levels of these lines are controlled by the **CDT** command. Some external ISDN and switch 56 TA's require that the DTR/CON line is asserted. The **CDT** command provides an easy method of controlling the DTR/CON line.

3.5 Encoder

Associated Remote Control Commands

EAD Calibrate AD converter
EAI Set encoder audio input source

EAL Set encoder algorithm
EAM Set encoder algorithm mode
EBR Set encoder bit rate
ELI Set encoder digital lines format
ESP Set scale factor protection
ESR Set encoder sampling rate

The compressed digital audio encoder is controlled by the above commands. If the decoder is dependent on the encoder (**DIN** NO), then some of these encoder commands also control the decoder.
The source of the audio input is controlled by the **EAI** command. The source may be the analog inputs or the digital AES/EBU inputs. The analog input AD converter is calibrated by the EAD command. This calibration is done at power-up but can be done at any time. The calibration process removes the effect of any DC voltage offset present at the input of the AD converter. This has a minor positive effect on the audio compression algorithm.
The encoder audio compression algorithm is set by the **EAL** command. If the algorithm is one of the ISO/MPEG types, then the EAM command set the mode to mono, dual mono, joint stereo or stereo. The digital audio sampling rate is controlled by the **ESR** command while the compressed audio bit rate is controlled by the **EBR** command.
The **ELI** command is used to control how the compressed digital audio bit stream is transmitted. For example, if **ELI** L1 is used, then the the compressed output bits are sent out digital interface (DIF) 1.
Scale factor protection (**ESP**) is used for ISO/MPEG types of bitstreams. Scale factors are the levels of the digital audio signal within a sub-band. There are 32 sub-bands and the scalefactors change the level over a 120 dB range. An error on any scale factor will cause a preceptable impairment in the audio. To prevent this, scalefactor protection can be inserted at the encoder and if the decoder is capable of recognizing it, then the decoder can perform a concealment operation to repair the damage scalefactor. If the decoder does not know about scale factor protection, the the audio is decoded and any damaged scalefactors cause an impairment. If **ESP** has enabled scalefactor protection, the far end decoder must enable scale factor correction by the **DSP** command.

3.6 Encoder Header

Associated Remote Control Commands

ECR Set encoder copyright bit in header
EEP Set encoder emphasis bit in header
EOR Set encoder original bit in header
EPR Set encoder protection bit in header

When utilizing the CCSO, CCSN or MPEG audio compression algorithm, there are certian flags which may be set in the header. These bits can be used by the decoder. These bits are defined below and the command used to set the bit is shown in parenthesis.

- Copyright (**ECR**)
- Emphasis (**EEP**)
- Original (**EOR**)
- Protection (**EPR**)

The cdqPRIMA decoder reads these bits and displays them. The state of these status bits can be seen by executing the **DRS** or the **CST** command.

3.7 Front Remote Control

Associated Remote Control Commands

CFB	Set front panel remote control baud rate
CFP	Set front panel remote control protocol usage
CFE	Set front panel remote control command response echo

Front panel remote control is provided on all models except the 110 and 210. Front panel remote control allows computer access to all the internal functions of the cdqPRIMA. Front panel remote control is especially useful for applications which need quick access to the cdqPRIMA via a palm top computer. This frequently occurs in control rooms in which there are many cdqPRIMA's in equipment racks.

The baud rate of the front panel access is set by the **CFB** command.

The protocol for this interface is defined by the **CFP** command. There are two possible protocols for communication with the cdqPRIMA. This first is simple ASCII messages which can be generated by any terminal emulator communications package. The second method of communications is via protocol protected messages. In this case, the simple ASCII message is surrounded by a header at the beginning of the message to specify the byte length of the message and other parameters and a CRC is appended to the end of the message for error control. The details of the protocol is covered in the chapter entitled cdqPRIMA Remote Control Protocol.

When downloading the cdqPRIMA, it is possible to turn off the command echo. This speeds up the download process at the expense of seeing the command echo. The command echo can be turned off by utilizing the **CFE** command.

3.8 Headphones

Associated Remote Control Commands

CHV	Set headphone volumn level of current device
DHV	Set decoder headphone volumn level
EHV	Set encoder headphone volumn level
CHP	Set headphone audio source

The front panel headphone output can be connected to either the encoder input signal (after the A/D converter) or to the decoder output (before the D/A converter) by the **CHP** command. The headphone can listen to the stereo signal (left channel to left earphone and right channel to right earphone) or the left channel only (left channel to left and right earphone) or the right channel only (right channel to left and right earphone).

The headphone volumn may be adjusted by the **CHV**, **DHV** and **EHV** commands. The volumn of the encoder and decoder are adjusted seperately. There are not separate adjustments for the left and right channels. The volumn level is from 0 to 127 arbitrary units with 0 being mute and 127 being the loudest.

3.9 Help

Associated Remote Control Commands

CQQ	Print command summary for common commands
DQQ	Print command summary for decoder commands
EQQ	Print command summary for encoder commands
HELP	Print all help commands
MQQ	Print command summary for maintenance commands

There are 4 categories of commands. These are

- Common commands
- Decoder commands
- Encoder commands
- Maintenance commands

Executing CQQ, DQQ, EQQ or MQQ lists a command summary for each of the command groups.

The commands are arranged in functional groups and these groups are displayed by executing the **HELP ?** command. A summary of each command group is shown by executing **HELP xx** where **xx** is a number between 1 and 30.

Each command has its own help. This help is displayed by typeing **HELP cmd** or cmd **HELP** where **cmd** is any three character command.

3.10 Hot Keys

Associated Remote Control Command

CHK	Define hot key

On certian models (the 230), user definable hot keys are available. These keys allow the user to attach a cdqPRIMA remote control command to a key. Once the command has been attached to the key, a depression of the key causes the command to execute. See the **CHK** command for a detailed explaination of the syntax of this command.

3.11 Loop Back

Associated Remote Control Commands

CBR	Set loopback bit rate
CLB	Set loopback on a digital data interface
CSL	Set system loopback

The cdqPRIMA has two types of loopback. The first type is a system loopback and the second is a digital interface loopback. The system loopback is an internal loopback and is set by the **CSL** command. It loops all the digital interfaces internally with one command.

The **CLB** command is used to set the loopback on each digital interface module. Some modules such as the X.21 and the V.35 card respond to this loopback. The TA cards generally do not respond to the **CLB** command.

When the **CSL** command is set to loopback (LB) then the internal clock is used as to supply the digital data clocks. The clock rate of this clock is set by the **CBR** command. The bitrate set by this command only applies when **CSL** is set th LB. When **CSL** is set to LB, the **EBR** and the **DBR** commands are ignored.

3.12 Maintenance

Associated Remote Control Commands

CDF	Set default parameters
MCP	Set connect port
MSY	Synchronize RAM and BBM
MVN	Print software version numbers
MWP	Set watch port

All of the cdqPRIMA parameters can be set to the factory default state by executing the **CDF** command. The psychoacoustic parameters and the speed dial numbers are not reset by the **CDF** command. The **CDF** command is also executed at power up when the 0 key on the front panel is depressed until the TOTAL RESET OF ALL PARAMETERS is displayed.

The **MCP** is used to connect the remote control port to an internal uart and monitor traffic to and from the specified serial port. It is used for debugging only and should be used only with the guidance of experienced technical support personal.

When commands are executed, the command argument is written to non-volatile RAM. For example if the **ELI** L1 command is issued, then the L1 is remembered in non-volatile RAM and if power is removed, the setting is remembered. When power is restored, the **ELI** L1 command is read from non-volitle RAM and executed in an attempt to restore the cdqPRIMA to the state that existed befor power was removed. Some commands write their argument to a cache which is later written to non-volitle RAM. The execution of the **MSY** command causes all entries to be written to non-volitle RAM immediately. This should be done just before powering down to insure that all parameters are in non-volitle memory.

The **MVN** command can be used to print the version number of the various software modules. It also prints the module checksum and length.

The **MWP** command is used for software debugging only. It should be used under the direction of an experienced maintenance technician.

3.13 Out Of Frame Detector

Associated Remote control Commands

MOC	Display OOF counter
MOD	Set OOF down count rate
MOL	Set OOF count rate limit
MOR	Reset OOF counter
MOU	Set OOF up count rate

The out of frame detector provides a method of monitoring the number of framing errors that occurred in the digital transmission path. The out of frame rate detector is used when ISO/MPEG type of frames are enabled by the **EAL** command. When each ISO/MPEG frame is received (every 24 milliseconds for 48 kHz sampling), the header CRC is checked for validity. If the frame header has valid framing bits, then the OOF counter is incremented by a OOF up count (any number from 0 to 9). If the frame header bits are invalid, then the OOF counter is decremented by the OOF down count (any number from 0 to 9).

If the OOF up count is set to 1 (by the **MOU** command) and the down count is set to 0 (by the **MOD** command), then the OOF counter counts the total number of frames in error. If the up counter is set to 2 and the down count is set to 1, then the OOF counter is sensitive to burst errors but not random errors.

The OOF counter is compared to the OOF threshold (set by the **MOL** command) to see if the counter is above or below the threshold. Actions such as closing a relay, dialing a phonenumber or lighting a LED or displaying a scrolling message can be taken.

The OOF counter can be reset to 0 by the **MOR** command.

The current contents of the OOF counter can be displayed by the **MOC** command.

3.14 Peak Detector

Related Remote Control Command

MPD	Display peak detector level

The **MPD** command is used to display the highest peak level for the encoder or the decoder, right or left channel. After executing this command, the highest peak level is set to -150 dBu and is updated by the the audio input. The peak level is retained even after all audio has stopped and can be read once by executing the **MPD** command.

3.15 cdqPRIMA Logic Language

Associated Remote Control Commands

CAR	Clear the latched value of the action word
CCT	Cancel timer

CEA Set event to action logic
CEV Print event inputs
CLA Print latched value of the action word
CRA Print realtime value of the action word
CTM Set timer timeout duration
CVA Define virtual action
ELU Set link message update rate
ESW Set a simulated switch

The cdqPRIMA has a rich language for mapping input events such as high BER into actions such as relay contact closure.

A detailed description of the cdqPRIMA Logic Language (PLL) is given in the cdqPRIMA Logic Language section.

The inputs to the Event to Action interpreter are displayed by the **CEV** command. These input events may be physical inputs such as input optical isolators or logical input such as computer generated switch closures (see the **ESW** command).

The mapping of input events into output actions is controlled by the CEA command. This command is described fully in the cdqPRIMA Logic Language section.

The real-time value of the Action Word is displayed by the **CRA** command while the latched value of the Action Word is displayed by the **CLA** command. The latched Action word values are reset via the **CAR** command.

Action Words are the output of the Event to Action logic which is controlled by the cdqPRIMA Logic Language (PLL). See the section entitled cdqPRIMA Logic Language for further details of the PLL. Actions are real and virtual. The real action are thing such as lighting a relay or virtual actions such as executing a remote control command (see the **CVA** command).

The other virtual action is the starting of a timer (see the **CTM** command). The expiration of a timer is an input event. Timers can be cancelled by the **CCT** command.

Actions can be exported to a far end cdqPRIMA. This exported action appears as an input event to the far end cdqPRIMA. The exported actions are transmitted to the far end at a rate governed by the **ELU** command. The actions are exported repeatedly in an attempt to insure their arrival at the far end even in the presence of a noisey digital communications channel.

3.16 Quiet Detector

Associated Remote Control Commands

MQC Display quiet detector level time left
MQD Display quiet detector level
MQL Set quiet detector level
MQT Set quiet time duration

There are 6 silence detectors (quiet detectors). These are

- encoder left channel input
- encoder right channel input
- encoder stereo input
- decoder left channel output
- decoder right channel output
- decoder stereo output

A stereo silence detector uses the greater of the left or right channel signal in the silence determination.

At .1 second intervals the audio levels of the encoder and decoder, left and right channels are measured. If the level is below the value set by the **MQL** command for a period of time set by the **MQT** command, then the channel is said to be silent. When a channel is silent, the silent event input is set to true. The value of the silence event may be used as input the the Event to Action logic interpreter.

The current value of any of the 6 quiet detectors can be displayed by the **MQD** command.

The time left before silence is detected can be displayed by the **MQC** command.

3.17 Psychoacoustic Parameter Adjustment

Associated Remote Control Commands

EPD	Get default psychoacoustic parameter table number
EPL	Load psychoacoustic parameters from flash
EPP	Set psychoacoustic parameter
EPS	Store psychoacoustic parameters in flash
EPT	Assign psychoacoustic parameter table
EPY	Set psychoacoustic parameter type

There are 32 psychoacoustic parameters which control the cdqPRIMA. The manipulation of these parameters is discussed in the sectio Psychoacoustic Parameter Adjustment.

3.18 Remote Control

Associated Remote Control Commands

CID	Set RS485 remote control ID
CPC	Set remote control protocol usage
CRB	Set remote control baud rate
CRI	Set remote control type
CRE	Set remote control command response echo

357

Rear panel remote control is provided on all models. Rear panel remote control allows computer access to all the internal functions of the cdqPRIMA. Rear panel remote control is especially useful for applications which need permenant access to the cdqPRIMA via a control computer. This frequently occurs when the cdqPRIMA is remotely located from the control room.

The rear panel remote control electrical interface may be either RS232 or RS485. The RS485 interface may be either the 2 or 4 wire interface. The choice of the electrical interface is controlled by the **CRI** command.

If protocol protected messages are used to control the cdqPRIMA, then the message must have a destination id. This id is set by the **CID** command.

The baud rate of the rear panel remote control port is set by the **CRB** command.

The protocol for this interface is defined by the **CPC** command. There are two possible protocols for communication with the cdqPRIMA. This first is simple ASCII messages which can be generated by any terminal emulator communications package. The second method of communications is via protocol protected messages. In this case, the simple ASCII message is surrounded by a header at the beginning of the message to specify the byte length of the message and other parameters and a CRC is appended to the end of the message for error control. The details of the protocol is covered in the chapter entitled cdqPRIMA Remote Control Protocol.

When downloading the cdqPRIMA, it is possible to turn off the command echo. This speeds up the download process at the expense of seeing the command echo. The command echo can be turned off by utilizing the **CRE** command.

3.19 Security

Associated Remote Control Command

CPW Set user's security status

CPW Set user's security status

3.20 Software Maintenance

Associated Remote Control Command

CVN Print software version number

The software version number of any flash object can be displayed via the **CVN** command. Each internal algorithm is called a Flash Object. Each Flash Object has its own internal version number.

3.21 Speed Dialing

Associated Remote Control Commands

CDS	Delete a speed dial number
CSC	Clear all speed dial entries
CSD	Speed dial a number
CSE	Enter a number in speed dial directory
CSF	Display first of speed dial entry
CSN	Display next of speed dial entry

The speed dial feature allows the entry of 256 system configurations consists of up to 6 telephone number, the sampling rate, line format and a description. Speed dial entries are entered by either the **CSE** command or the **SDSET** button. Each of the speed dial entries is given a speed dial id (3 digit number). A speed dial entry is activated by either the execution of the **CSD** command or the front panel **SDIAL** button.

All speed dial entries can be deleted via the **CSC** command while a single speed dial entry can be deleted by the **CDS** command.

The entire s· eed dial list may be displayed by first typing **CSF** to display the first speed dial entry and then entering **CSN** repeatedly to display the subsequent speed dial entries. The speed dial entries are displayed in alphabetical order by description.

3.22 Status and Level Display

Associated Remote Control Commands

CLI	Set LED display intensity
CLM	Display LED message
CVU	Set level meter mode

A front panel LED display is provided on all models except the 110 and the 210. This front panel display can be used for various functions. The CVU command is used to set the measurement mode. The normal mode is the level indication mode in which the average and peak input signal is displayed. The stereo image can be displayed as well as the left/right channel corrolation.

The level mode of operation is the usual level indicator mode. The right hand side of the level display is labeled 0 dB and each LED to the left represents 2 dB weaker signal. The right hand 5 LED's are red, the next 5 LED's to the left are yellow and the last 10 LED's on the left are green. Thus the 20 LED's represents a 40 db range.

The far right LED has a reversed arrow display to indicate that the input or output is at the maximum level of 0 dB. The VU meter is labeled with 0 a sthe maximum because the input amplifiers may be different values. For example, the standard input amplifier on the cdqPRIMA allows a maximum input of +18 dBu. If a sinewave with a peak to peak level of +18 dBu is input to this amplifier module, the peak level LED will read 0. A 0

level of the level LED means that the input is at the maximum allowed value. The output LED display is similar.

The level display consists of and encoder and a decoder section, each with a left and a right channel display. Each channel display consists of a single LED representing the peak value and solid group of LED's representing the average value of the input audio.

If the stereo image display is selected, the the scale below the display must be used. This scale shows the relative location of the stereo image. If the image is centered, then the single LED is illuminated above the C. If the image is to the right then the LED is displayed to toward the L. This display is useful when the gains of the left and right channels must be balanced for stereo signals.

The stereo corrolation display is indicated by a double LED illumination. The corrolation display is useful to detect if the input signal can be mixed to mono. A corrolation from 0 to + 1 indicates that there is mono compatibility while a stereo corrolation near -1 indicates that the left and right signals are out of phase and cannot be mixed to mono.

The **CLM** command allows a scrolling message to be displayed on the front panel LED display. This is useful to alert a remote location of an upcomming feed or provide a cue.

The **CLI** command is used to set the intensity of the LED display. The display is broken into 3 groups and the intensity of each group can be controlled. This allows instant focus on one group by dimming the intensity on the other groups.

3.23 Status

Associated Remote Control Command

CST Report CODEC status

A general system status is provided when the **CST** command is executed. This status is intended to be a snapshot of all system functions.

3.24 System Setup

Associated Remote Control Command .

CDF Set default parameters

The **CDF** command is use to restore the factory defaults for everything except the psychoacoustic parameters and the speed dial numbers. To test the unit that seems to be confused, one can issue the **CDF** followed by the **CSL LB** commands to set the defaults and set the system into loopback. See the CDF command for a list of the system defaults.

3.25 Terminal Adaptor

Associated Remote Control Commands

CAA	Set TA auto answer mode
CAC	Set TA auto-reconnection state
CAD	Auto dial phone numbers
CCR	Clear TA digital interface connect time
CCS	Print TA digital interface connect time
CDC	Real-time display TA digital interface connect time on LCD
CDI	Dial TA phone number
CHU	Hang up a line or lines
CLD	Set ID for a Terminal Adaptor
CSI	Set SPID for a Terminal Adaptor
CSW	Set switch type
CTC	Connect to a TA control port
CTE	Set TA remote control command response echo
CTP	Set TA remote control protocol usage
CTO	Set TA dialing timeout

The ISDN type of digital interface module allows access to the ISDN network. There are several types of ISDN TA's available for the cdqPRIMA.

The TA101 provides 1 BRI (2 * 64 kbs) access to the network. This TA requires different ROMS for different countries. The TA201 and TA202 ROMS have onboard FLASH memory with the switch configurations for different countries. Contact the factory to obtain the proper ROM for your country if you are utilizing a TA101 TA.

If the TA is operated in North America, the the switch type (**CSW**), line ID (**CLD**) and line SPID (**CSI**) must be entered before any calls can be placed. See Appendix A for TA101 setup information.

A direction connection with the TA is performed by the **CTC** command. This mode of operation is useful because it allows the lowest level of control over the TA. When the CTC command is used, then all of the low level TA commands are available. Consult the factory for a description of these low level commands.

The **CAA** command can be used to set the TA into the auto answer mode. If the TA is not in the auto-answer mode, then it will not accept any incomming calls.

An indivudual line may be connected by utilizing the **CDI** command. This command allows dialing individual ISDN lines at either 56 or 64 kbs. Once a call has been placed to the farend, a timeout is in effect waiting for the far end to answer. This timeout is set by the **CTO** command.

Once a call has been placed, by the **CSD** or **CDI** command, the line or lines may be "hung up" by the **CHU** command. This command disconnects a connected line.

If a connection is made to a far end TA and the connection is lost, it is possible to have the cdqPRIMA automatically re-establish the connection. This is done by the **CAC** command.

The cdqPRIMA allows the display of the time the line has been connected of any one of the 6 digital interfaces. This is useful for estimating the cost of the connection. The **CDC** command is used to display the connect time on the LCD screen. The current time connected for any of the 6 lines can be printed on a remote control terminal by the **CCS** command. The connect time counter can be set to zero at any time by the **CCR** command.

The cdqPRIMA allows direct connection over ISDN into the ISDN remote control port. This allows complete remote control including software down load from a far end cdqPRIMA via ISDN. The **CTP** command is used to enable or disable command protocol usage over the ISDN line while the **CTE** command us used to control the command response echo.

3.26 Test

Associated Remote Control Commands

MTM	Perform a test measurement
MET	Enable hardware tests

MET Enable hardware tests

The cdqPRIMA can be used to perform various tests on external equipment. These tests are controlled by the MTM command.

3.27 Time Code

Associated Remote Control Commands

CTI	Set Time Code readout source
CTL	Print last Time Code received
CTS	Print Time Code speed
CTT	Enable/disable Time Code

SMPTE time code is an optional feature of the cdqPRIMA. SMPTE time code is read by the optional reader, converted into a digital bit stream, muxed with other data and send to the decoder as ancillary data. The mux mode of ancillary data (**CAN 2**) must be used and the audio algorithm cannot be G.722 in order to use SMPTE timecode.

The SMPTE timecode reader and generator in the cdqPRIMA automatically sense the the input timecode rate with no external control necessary. The cdqPRIMA allows the user to transmit timecode simultaneously with the audio and thus the cdqPRIMA is the perfect unit for studios utilizing audio/video timecode.

SMPTE timecode utilizes approximately 2.4 kbs of digital bandwidth. This small overhead allows transmission of timecode even at bits rates of 56 and 64 kbs. If the timecode input is removed, then no digital bandwidth is used. It may be inconvenient to remove the timecode input and the **CTT** command can be used to enable/disable the transmission of timecode. Turning timecode off with the **CTT** command has the same effect as removing the timecode connector from the rear of the cdqPRIMA.

The **CTS** command is used to print the timecode speed.

The current time code may be displayed by the **CTI** command. The displayed timecode may be the timecode input to the encoder or the timecode received by the decoder.

The last timecode received may be displayed by the **CTL** command.

3.28 Timing

Associated Remote Control Commands

DES	Decoder AES timing
ETI	Encoder timing
DDO	Set digital output sampling rate
DTI	Decoder timing

The timing of the encoder and decoder can be contolled by various commands. These commands are documented in the Digital Timing Section of this manual.

3.29 Misc

Associated Remote Control Command

COM Comment command

The **COM** command performs nothing and is useful for inserting comments in command scripts.

3.30 Download/Boot

Associated Remote Control Commands

MBM Boot the cdqPRIMA from ROM

Normally the cdqPRIMA executes its software from the FLASH memory. If this FLASH memory needs to be updated, the it must operate out of the the boot ROM.

The **MBM** command is used to force the cdqPRIMA to operate from the ROM boot. This is required when downloading new software into the cdqPRIMA.

3.31 Sync Ancillary Data

Associated Remote Control Commands

DSB	Set decoder synchronous ancillary data rate
DSC	Set decoder synchronous ancillary data clock edge
ESB	Set encoder synchronous ancillary data rate
ESC	Set encoder synchronous ancillary data clock edge

The synchronous ancillary data commands allow the bit rate (**DSB** and **ESB**) to be set for the decoder and encoder. The clock edge (low to high or high to low) for clocking valid data can also be set for the encoder and the decoder (**DSC** and **ESC**).

4. Operation

4.1 Quick Start

The cdqPRIMA is shipped from the factory configured for loopback operation. This means that at power up, the cdqPRIMA should operate correctly and pass audio from the input to the output. The settings for the encoder are given under the **CDF** command but they are summarized below.

parameter	*value*
bitrate	128 kbs
algorithm	MPEGL2
mode	joint stereo
sampling rate	128
encoder line format	L1
decoder set to independent	NO

Table 4-1

Summary of default setups

4.2 Front Panel Displays

4.2.1 Character Display (Models 110, 120, 210 & 220)

The LCD display for the cdqPRIMA models 110, 120, 210 and 220 models is a 2 line by 16 characters. This display is used for all responses to front panel user commands as well as spontaneous messages such as incoming call connect messages.

4.2.2 Graphics Display (Model 230)

The cdqPRIMA model 230 has a graphics display which allows 8 rows of 40 characters or 240 by 64 pixels. When operating in the character mode, the display functions in a manner similar to the cdqPRIMA 110. The graphics mode is used for graphical display of measurement information.

4.3 Front Panel Controls

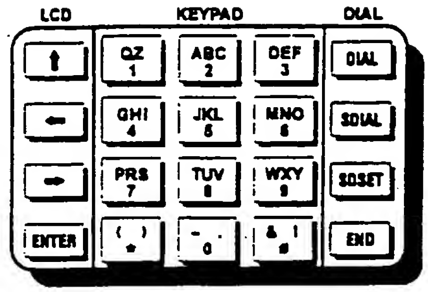


Figure 1

Model 110 & 210 keypad

4.3.1 Cursor Keys (All Models)

The 4 keys under the LCD label are used to control the cursor. They are

- UP ARROW
- LEFT ARROW
- RIGHT ARROW
- ENTER

The up arrow key is used to move up the menu tree. This key is also use on power up to force entry into the ROM boot mode which is used for local downloading software. The up arrow key is also used to terminate any graphical measurements which are in progress.

The left and right arrow keys are used to move to the right and left in the menu tree.

The **ENTER** key is used to execute the menu tree entry enclosed within the square brackets ([]).

4.3.2 Dial Keypad (All Models)

The dial keypad consists of the 12 keys under the KEYPAD label. These keys forma a general purpose alpha-numeric keypad. Different commands enable different characters on these keys. For example, dialing commands only enable the numeric selections for these keys. When cdqPRIMA Logic Language commands are entered, all of the keys are enabled. By depressing the 2 key repeatedly, the A, B and C keys are displayed. In such multi-character modes are enabled, the right and left arrow keys are used to move to the right and left on the current line. The Enter key is used to accept the entire entry.

4.3.3 Dial Setup Keys (All Models)

The 4 keys below the **DIAL** label are used for dialing. They are

- **DIAL**
- **SDIAL**
- **SDSET**
- **END**

The dial key allows the dialing of a single ISDN line. Before dialing can be attempted. the Digital InterFace (DIF) must be defined by utilizing the CIF command. The DIF must contain a TA type of Digital Interface Module (DIM) such as a TA101.

Depressing the **DIAL** key begins the dialing sequence and the LCD display will prompt the user for the bit rate and telephone number. Once the enter key is depressed denoting the entry of the phone number, then the dialing operation begins and the DIF LED begins to blink indication that the phone is dialing. When the light becomes solidly on, the connection is established. The calling status is also displayed on the LCD screen.

The **SDIAL** key is used to speed dial a destination. After depressing **SDIAL**, the LCD screen prompts for the 3 digit speed dial number which is terminated by depressing the **ENTER** key. The parameter required by this operation is described in the CSD remote control command.

The **SDSET** key is used to setup a speed dial entry. Depressing this key produces a series of prompts on the LCD display to enter the speed dial parameters. The parameters to be entered are described in the CSE remote control command.

The **END** key is used to terminate a connections made by the **DIAL** and **SDIAL** keys. Depressing this key allows all lines or a single line to be dropped. See the CHU command.

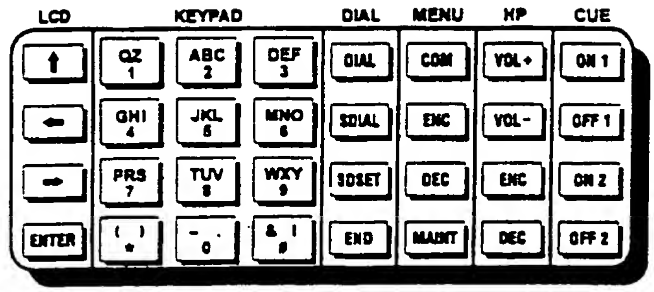


Figure 2

Model 120 & 220 keypad

4.3.4 Menu Keys (Models 120, 220 & 230)

The 4 keys under the MENU label are used to quickly move the one of the 4 main branches of the menu tree. These branches are

- COM Commands common to the entire unit
- ENC Commands for the encoder
- DEC Commands for the decoder
- MAIN Maintenance commands

4.3.5 Headphone Keys (Models 120, 220 & 230)

The 4 keys under the HP label are used to control the output of the front panel headphone jack. These keys are

- **VOL+**
- **VOL-**
- **ENC**
- **DEC**

The keys labeled **ENC** and **DEC** are used to select the encoder and decoder respectively. If the **ENC** button is depressed, the input signal to the encoder section is output to the headphone. If the **DEC** button is depressed, the decoder output is present at the headphone jack. There are 4 LED's under the label **HP STATUS** which are controlled by the **ENC** and **DEC** push buttons. If the **ENC** button is depressed, one or both of the encoder headphone LED's illuminate. When the **ENC** button is first depressed, the output of the left and right channels are output to the left and right earphones. If the ENC

button is depressed again, the encoder left channel LED is illuminated and the input to the encoder left channel is output to both the left and right channel headphones. If the ENC button is depressed again, the left channel HP LED is illuminated and the signal which is input to the right channel of the encoder is connected to both the left and the right channel of the headphones. A similar action occurs when the **DEC** button is repeatedly depressed.

The **VOL+** and **VOL-** buttons control the volume of the headphone output. Depressing the **VOL+** increases the headphone volume while depressing the **VOL-** button decreases the headphone volume. The headphone volume level ranges from 0 (mute) to 127 in arbitrary volume units (approximately 1 dB steps).

The volume buttons control the left and right channels simultaneously but the encoder and decoder output signals have separate volume levels which are active when the ENC and the **DEC** buttons are depressed.

If the headphone volume is set too high, distortion may occur.

4.3.6 Cue Keys (Models 120, 220 & 230)

The 4 buttons under the CUE label are general purpose front panel switches. These 4 buttons represent 2 switches. These two switches can be either on or off. For example, depressing the **ON1** button, causes switch 1 to turn on while depressing the **OFF1** button causes switch 1 to turn off. The corresponding action occurs for the **ON2** and **OFF2** buttons for switch 2. Switch 1 and switch 2 are connected to CI1 and CI2 (see the section on cdqPRIMA Logic Language).

The default setup of the cdqPRIMA assigns switch 1 (**ON1** and **OFF1** buttons) to a sending a cue from the near end to a far end. Depressing the **ON1** button, causes the SCUE1 LED to illuminate indicating that cue 1 is being sent. The far end cdqPRIMA will illuminate its RCUE1 LED indicating that it received this cue. Depressing the OFF1 button causes the SCUE1 LED to extinguish indicating that there is no cue 1 being sent.

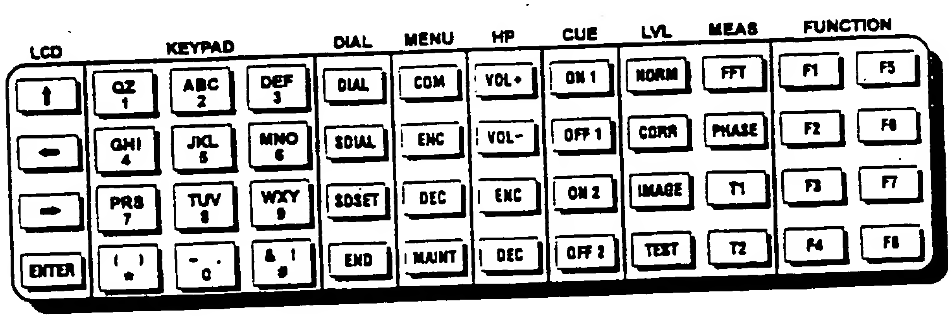


Figure 3

Model 230 keypad

BAD ORIGINAL

4.3.7 Level Control Keys (Model 230)

The 4 keys below the LVL label on the front panel are used to control the audio level LED display. The keys are labeled

- **NORM**
- **CORR**
- **IMAG**
- **TEST**

Depressing the **NORM** key causes the audio level LED's to display the average and peak levels. Each LED represents 2 dB and the signal corresponding to the maximum input is labeled 0 dB.

Depressing the **CORR** key causes the level LED's to display the stereo correlation. The values for the left/right correlation are +1 to -1 where +1 indicates the left and right channels are exactly in phase. A correlation of -1 indicates that the left and right channels are exactly out of phase. In phase stereo signals may be mixed into a mono signal.

Depressing the **IMAG** key causes the level LED's to display the stereo image of the left and right channel. If the power of the left and right channels are the same, then the stereo image will be in the center above the stereo image label C. If the power of the right channel is more than the left channel, the stereo image LED will move to the right indicating the stereo image has moved to the right.

Depressing the **TEST** button causes all the LED's to illuminate for a few seconds to allow visual inspection of all the LED's.

4.3.8 Measurement Keys (Model 230)

The 4 measurement keys

- **FFT**
- **PHASE**
- **T1**
- **T2**

are use for graphics measurements. The results of all these measurements are displayed on the graphics display.

The key labeled **FFT** is used to enable the real-time spectrum analyzer of the signal which is input to the left channel of the encoder.

The **PHASE** key is used to display a real-time phase display of the left and right channels.

The **T1** and **T2** keys are currently not assigned to any measurement function.

4.3.9 Function Keys (Model 230)

The 8 keys labeled **F1** through **F8** are user definable function (hot) keys. Any remote control command may be attached to any of these keys. See the **CHK** command for instructions on how to define one of these hot keys.

4.4 Front Panel Indicators

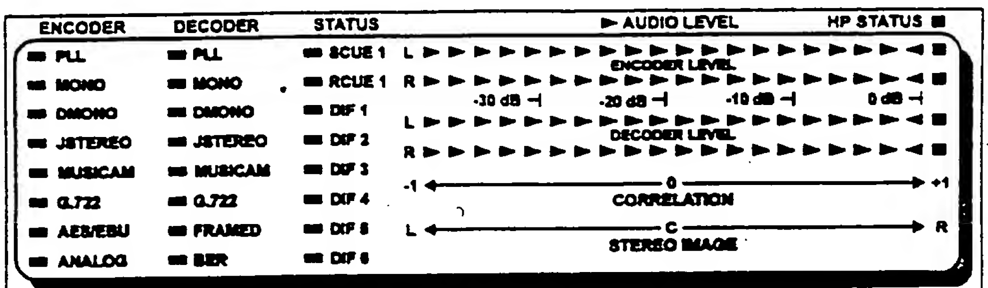


Figure 4

Model 120 and 220 Level Display

4.4.1 Model 120 and 220

4.4.1.1 Encoder

4.4.1.1.1 PLL

This LED illuminates green when the encoder phase locked loop is locked. This LED must be on for proper operation.

4.4.1.1.2 MONO

This LED illuminates yellow when the ISO/MPEG frame is transmitting a mono signal. This led is also illuminated when G.722 is being transmitted.

4.4.1.1.3 DMONO

This LED illuminates yellow when the ISO/MPEG frame is transmitting dual mono.

4.4.1.1.4 JSTEREO

This LED illuminates yellow when the ISO/MPEG frame is transmitting in the joint stereo mode. If the MONO, DMONO and JSTEREO LED's are all extinguished, then the encoder is outputting stereo frames.

4.4.1.1.5 MUSICAM

This LED illuminates yellow when CCS MUSICAM or ISO/MPEG frames are being transmitted.

4.4.1.1.6 G.722

This LED illuminates yellow when G.722 audio compression is being transmitted.

4.4.1.1.7 AES/EBU

This LED illuminates yellow when the input audio source is from the rear panel AES/EBU, SPDIF or optical inputs.

4.4.1.1.8 ANALOG

This led illuminates yellow when the input audio source is from the rear panel analog XLR connectors.

4.4.1.2 Decoder

4.4.1.2.1 PLL

This LED illuminates green when the encoder phase locked loop is locked. This LED must be on for proper operation.

4.4.1.2.2 MONO

This LED illuminates yellow when an ISO/MPEG frame is received and it is a mono signal. This LED is also illuminated when G.722 is being received.

4.4.1.2.3 DMONO

This LED illuminates yellow when an ISO/MPEG frame is received and its format is dual mono.

4.4.1.2.4 JSTEREO

This LED illuminates yellow when and ISO/MPEG frame is received and the type of frame is joint stereo. If the MONO, DMONO and JSTEREO LED's are all extinguished. then the decoder is receiving stereo frames.

4.4.1.2.5 MUSICAM

This LED illuminates yellow when CCS MUSICAM or ISO/MPEG frames are received

4.4.1.2.6 G.722

This LED illuminates yellow when G.722 audio compression are received.

4.4.1.2.7 FRAMED

This LED is used to indicate that the cdqPRIMA is receiving a properly framed signal. It illuminates green when the cdqPRIMA is framed.

4.4.1.2.8 BER

This LED is used to indicate that a bit error has been detected. This LED illuminates red when a bit error has been received.

4.4.1.3 Status

4.4.1.3.1 SCUE1

Normally this LED illuminates when the **ON1** button is depressed and is extinguished when the **OFF1** button is depressed. Its normal meaning is that a cue has been sent to a far end decoder to be displayed on the far end RCUE1 LED.

The SCUE1 LED can be programmed to mean other things. See the chapter entitled cdqPRIMA Logic Language. for programming instructions.

4.4.1.3.2 RCUE1

Normally this LED illuminates when cue 1 has been received from the far end encoder.

This LED can be reprogrammed to mean other things. See the chapter entitled cdqPRIMA Logic Language. for programming instructions.

4.4.1.3.3 DIF1, DIF2, DIF3, DIF4, DIF5 and DIF6

There are 6 LED indicators for the digital interface status. These LED's can be in 3 states. These are

- OFF (disconnected)
- BLINKING (TA dialing)
- ON (connected)

These states corresponding the interface status in parenthesis.

4.4.2 Model 230

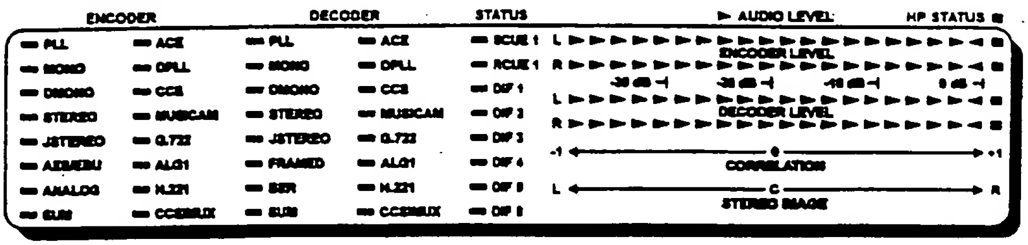


Figure 5

Model 230 Level Display

4.4.2.1 Encoder

4.4.2.1.1 PLL

See Model 120 display for a description.

4.4.2.1.2 MONO

See Model 120 display for a description.

4.4.2.1.3 DMONO

See Model 120 display for a description.

4.4.2.1.4 STEREO

This LED illuminates yellow when an ISO/MPEG type of frame is sent and the mode of the signal is stereo.

4.4.2.1.5 JSTEREO

This LED illuminates yellow when an ISO/MPEG type of frame is sent and the mode of the signal is joint stereo.

374

4.4.2.1.6 AES/EBU

See Model 120 display for a description.

4.4.2.1.7 ANALOG

See Model 120 display for a description.

4.4.2.1.8 SUM

This LED illuminates when the encoder has detected an error. This LED is programmable and thus its meaning depends on the current definition. See the chapter entitled cdqPRIMA Logic Language.

4.4.2.1.9 ACE

This LED illuminates yellow when ACE (Advanced Concealment of Errors) is enabled. ACE protects sensitive parts of the audio frame in the presence of bit errors. The decoder must have ACE enabled for the reduction to bit errors to be effective.

4.4.2.1.10 DPLL

This LED illuminates yellow when the encoder AES/EBU, SPDIF or OPTICAL digital audio input is present. An illuminated DPLL LED is required for proper operation of digital audio input signals.

4.4.2.1.11 CCS

This LED illuminates when an older CCS type of compressed audio frame is transmitted. This LED should be illuminated for proper interoperation with older CCS CDQ20xx decoder to insure proper operation at all bit rates.

4.4.2.1.12 MUSICAM

This LED illuminates yellow when an ISO/MPEG or new CCS compressed audio frame is transmitted. If this LED is illuminated, the cdqPRIMA will interoperate with any ISO/MPEG layer 2 compliant decoder.

4.4.2.1.13 G.722

See Model 120 display for a description.

4.4.2.1.14 ALG1

Currently not used.

4.4.2.1.15 H.221

This LED illuminates yellow when J.52 type of H.221 multiple bonding is in effect.

4.4.2.1.16 CCSIMUX

This LED illuminates yellow when the CDQ20xx type of 2 line (2x56 or 2x64) bonding is in effect.

4.4.2.2 Decoder

4.4.2.2.1 PLL

See Model 120 display for a description.

4.4.2.2.2 MONO

See Model 120 display for a description.

4.4.2.2.3 DMONO

See Model 120 display for a description.

4.4.2.2.4 STEREO

This LED illuminates yellow when an ISO/MPEG type of audio frame is received whose mode is stereo.

4.4.2.2.5 JSTEREO

This LED illuminates yellow when an ISO/MPEG type of audio frame is received whose mode is stereo.

4.4.2.2.6 FRAMED

See Model 120 display for a description.

4.4.2.2.7 BER

See Model 120 display for a description.

4.4.2.2.8 SUM

This LED illuminates when the decoder has detected any error condition. This LED is programmable and thus its meaning depends on the current definition. See the chapter entitled cdqPRIMA Logic Language

4.4.2.2.9 ACE

This LED illuminates yellow when the decoder is set to expect ACE type of frame protection. ACE reduces the sensitivity of the compressed audio to bit errors. If the decoder has ACE enabled, the far end encoder must also have ACE enabled. If the

decoder has ACE enabled and the far end does not have ACE enabled, then the decoder will mute.

4.4.2.2.10 DPLL

This LED illuminates yellow when the decoder AES/EBU, SPDIF or OPTICAL sync input is receiving a valid digital sync signal. This signal must be present if decoder digital audio synchronization is required.

4.4.2.2.11 CCS

This LED illuminates when the decoder is receiving an older version of the CCS MUSICAM.

4.4.2.2.12 MUSICAM

This LED illuminates yellow when the decoder is receiving either ISO/MPEG compliant frames or new CCS MUSICAM compressed digital audio frames.

4.4.2.2.13 G.722

See Model 120 display for a description.

4.4.2.2.14 ALG1

See Model 120 display for a description.

4.4.2.2.15 H.221

This LED illuminates yellow when J.52 type of H.221 multiple bonding is in effect. The far end encoder must be utilizing J.52 type of bonding if the decoder is in the J.52 mode.

4.4.2.2.16 CCSIMUX

This LED illuminates yellow when the CDQ20xx type of 2 line (2x56 or 2x64) bonding is in effect. The far end encoder must be utilizing the CDQ20xx type of 2 line bonding if the decoder is in the CDQ 2 line mode.

4.4.2.3 Status

All these displays identical to the status displays on the Model 120.

4.4.3 Level LED's (Model 120, 220, 230)

4.4.3.1 Peak & Average Level Indications

The level mode of operation allows the average level and the peak level of the signal input to the encoder and the signal output from the decoder. Each LED represents 2 dB

of signal level and the maximum level is labeled 0 dB. This maximum level is highest level permissible at the input or at the output of the cdqPRIMA. All levels are measured relative to this maximum level. The level LED's display a 40 dB audio range.

The peak hold feature of the level LED's shows the highest level of any audio sample. This value is instantly registered and the single peak level LED moves to the value representing this signal. If the peak level of all future signals are smaller, then the peak level led slowly decays to the new peak level. The peak level LED has a fast attack and a slow decay..

4.4.3.2 Stereo Image Display

The stereo image display is used to display the position of the stereo image. This is useful when setting the levels of the left and right channels to insure the proper balance.

4.4.3.3 Correlation Display

This display is used to check if the left and right channels are correlated (+1). If the left and right channels are correlated, then they can be mixed to mono.

4.4.3.4 Message Display

The level LED's can be used to display a scrolling message.

4.4.3.5 Selective Dimming

The Status, Encoder and Decoder groups of LED's can be independently dimmed to allow emphasis of a particular group.

4.4.4 Headphone Status Indicators (Model 120, 220 & 230)

The headphone indicators at the far right of the level displays are used to denote the signal output to the headphones. If both LED's are illuminated, then the left channel is output to the left earphone and the right audio channel is output to the right earphone. If only the left LED is illuminated, the left audio channel is output to both the left and right headphone. Similarly if the right channel headphone LED is illuminated.

4.5 Front Panel Connectors

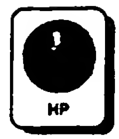


Figure 6

Headphone Jack

4.5.1 Headphone Jack (Model 120, 220 & 230)

The front panel 1/4 inch headphone jack is located on the front panel for convenient monitoring of input or output signals. The level and control of the headphone output is controlled by the front panel push buttons under the HP heading or by remote control commands.

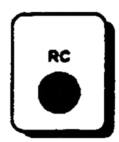


Figure 7

Remote Control

Port

4.5.2 Front Panel Remote Control Port (Model 120, 220 & 230)

The front panel remote control port is used to control all internal operations of the cdqPRIMA. It has the same functionality as the rear panel remote control connector.

4.6 Power Up Boot Sequence

At system power up, the cdqPRIMA loads the control processor and the various DSP's (Digital Signal Processors) from FLASH memory. It perform various power on checks and then starts execution of all its sub-systems.

4.7 System Setup

4.7.1 Menu Operation

The LCD menu sub-system is arranged like a tree. The up, right, left and enter keys allow navigation through the tree. See the chapter entitled LCD MENU TREE SUMMARY for the details of the menu tree.

4.7.2 Speed Dial Loading

4.7.2.1 Manual Loading

4.7.2.2 Via Remote Control

4.8 Digital InterFace (DIF) Setup

Before any connection to the outside world is possible, the digital interface modules in the cdqPRIMA must be defined. They will be set at the factory but if the Digital Interface Modules (DIM's) are rearranged, then the cdqPRIMA must be notified. This notification is done by the CIF remote control command or the Define I/F on the LCD menu.

There are 2 basic types of interfaces and these are TA (terminal adapter) and non-TA types (X.21, RS422, RS485).

There is one slot in the 1xx series models and there are 3 slots in the 2xx series. The slots are numbered

- DIF12
- DIF34
- DIF56

and are associated with digital interface 1 and 2 for DIF12 and so on.

In the future, the DIF12 slot will be expanded to include DIF34 as well.

4.9 Dialing with Internal ISDN Terminal Adapter(s)

4.9.1 General Dialing and Auto Reconnect

The cdqPRIMA has two methods of dialing. They are single line dialing and multiple line dialing (speed dialing). For either mode of dialing, it is possible to enable automatic reconnect. This feature allows the automatic reconnection of a dropped line. If auto reconnect is enabled (see the CAC command) when a line is dialed, then it will be reconnected if either the far end disconnected the call or the network drops the call. If the calling end drops the call, the the line will not be automatically reconnected.

4.9.2 Manual Dialing

Dialing a number with the cdqPRIMA is straightforward. Depress the DIAL button on the front panel and enter the DIF, the digital data rate (56 or 64) and the far end phone number. The left and right arrow keys are used to make the proper selection. The ENTER key is depressed to activate each selection. While the cdqPRIMA is dialing, the front panel LED (on the units with LED's) blinks for the DIF that is dialing. When the line is connected, the DIF LED illuminates with a steady light.

4.9.3 Hanging Up a Connection

The cdqPRIMA allows the disconnection of individual lines or all connected lines. To initiate the disconnection process, depress the front panel button labeled **END**. Make the ALL or the individual line selection and depress the **ENTER** button to disconnect the line or lines.

4.9.4 Speed Dial

Speed dialing requires that the speed dial configuration is entered. Assuming that this number is entered, then simply depressing the **SDIAL** button on the front panel followed by the entering of the speed dial ID and then depressing the **ENTER** button causes the far end number(s) to be dialed and the cdqPRIMA setup as required.

If speed dialing is used to establish the connection, the **END** key is used to terminate the call just like any other connection is disconnected.

4.10 Resetting to Factory Defaults

When the cdqPRIMA is first turned on, it goes through a power up sequence. The first stage of the boot process is the ROM boot and the second stage is the FLASH boot. At the end of the second stage of the boot process, the cdqPRIMA looks to see if one of 4 keys are depressed. Depressing one of the these keys has the following result.

- **1** reset all operational parameters (execute the CDF command)
- **2** erase all speed dial entries
- **3** set all psychoacoustic parameters to the factory default
- **0** all the above operations

The front panel button should be depressed until an acknowledgement of the depressed key is shown on the LCD screen. For example, if the 0 key is depressed during power up, then it should be held until the message

TOTAL RESET OF DEFAULT PARMS

appears.

5. Digital Timing

The timing of the digital sections of the encoder and the decoder are controlled by various cdqPRIMA remote control commands.

The decoder timing is derived in 4 different ways. These modes of timing are set by the **DTI** command and are

- NORMAUTO
- INTAUTO
- INT
- AES

Let us first examine the NORMAUTO mode of timing. In this mode of operation, the timing of the decoder output is directly connected to the DA converter and the AES/EBU transmitter for the sampling rates of 48, 44.1 and 32 kHz. For the sampling rates of 24, 22.04 and 16, the decoder output is rate adapted before it goes to the DA and the

Decoder Sampling Rate	*Rate Adaption Used*	*Output Sampling Rate*
48	NO	48
44.1	NO	44.1
32	NO	32
24	YES	48
22.05	YES	32
16	YES	29.5

Table 5-1

Decoder rate adaption

AES/EBU transmitter. The table below shows the configurations.

382

The block diagram of the timing is shown below for the non rate adapted and the rate adapted case.

For **DTI** set to INTAUTO, a rate adaptor is used in all cases. The operational table for this mode is shown below.

Decoder Sampling Rate	*Rate Adaption Used*	*Output Sampling Rate*
48	YES	48
44.1	YES	48
32	YES	48
24	YES	48
22.05	YES	32
16	YES	29.5

Table 5-2

Decoder rate adaption

If **DTI** is set to INT, then rate adaption is always used and the **DDO** command is used to set the output sampling rate. Care must be taken when utilizing the **DDO** command to set the sampling rate because not all combinations of rates are possible. See the **DDO** command for the table of possiblities.

If **DTI** is set to AES, then the output sampling rate is determined by the AES sync input The decoder sync input may or may not be available. The **DES** command is used to control the timing requirement for the sync input.

5.1 Decoder direct connect

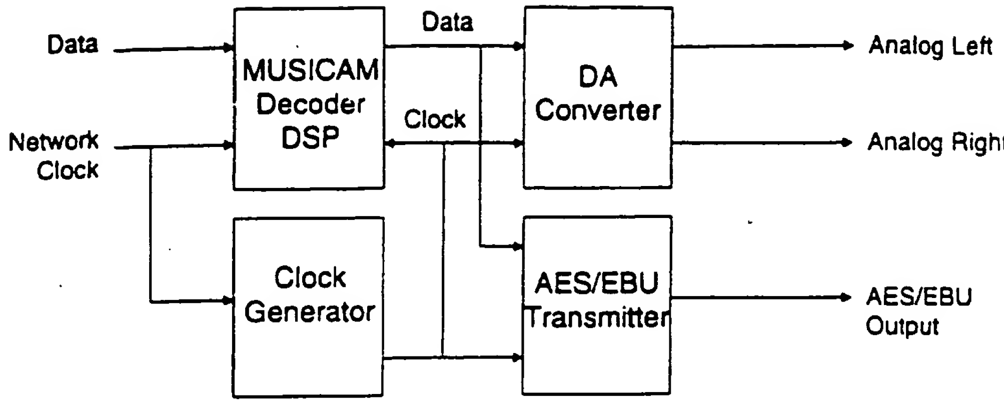


Figure 5-1

Decoder output timing with AES/EBU sync disabled or not present using normal timing

5.2 Decoder With Rate Adaption

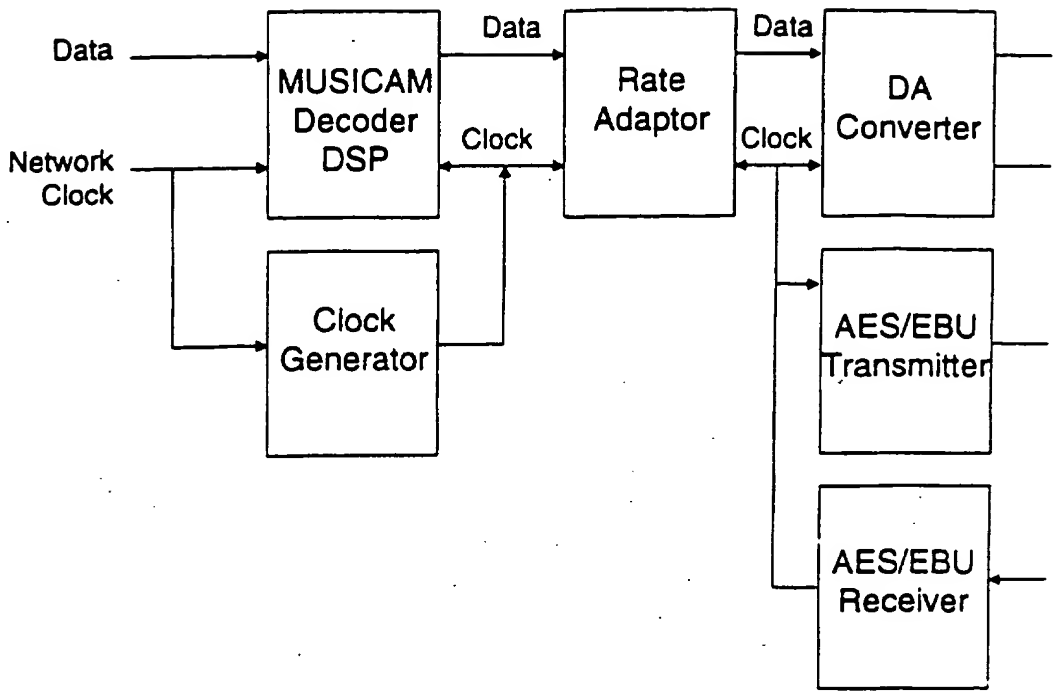


Figure 5-2

Decoder output timing with AES/EBU sync enabled and present using AES timing

5.3 Decoder with Rate Adaption

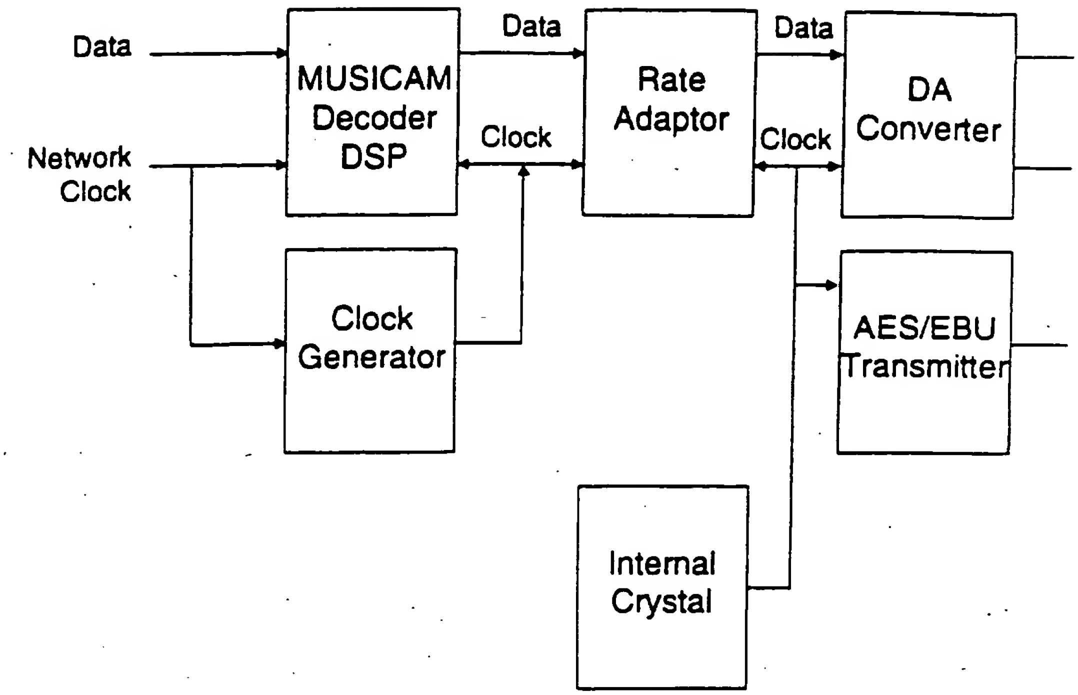


Figure 5-3

Decoder output timing with AES/EBU sync disabled or not present using internal crystal timing

6. Psychoacoustic Parameter Adjustment

There are 32 psychoacoustic parameters which control the cdqPRIMA. These parameters are numbered 0 through 31 and are set by the **EPP** command. Each of the 31 parameters can be of one of 4 types. These are dB, Bark, floating point and integer. The type of each parameter is set by the **EPY** command and should not be changed.

There are 20 different compressed digital audio bit rates and 6 sampling rates. This makes a total 120 different psychoacoustic parameter tables. The tables are numbered 0 to 239. The tables from 0 to 119 hold user defined parameters while the tables from 120 to 239 hold the factory defined tables. The tables from 120 to 239 should never be changed but they can be copied to a user defined table (0 to 119) and modified.

When the encoder is set to operate at a specified sampling rate and bit rate, the corresponding psychoacoustic table is loaded into the encoder. The current table number used for each sampling rate and bit rate can be displayed or changed by the **EPT** command.

To modify a factory default table, store it in a user table and tell the encoder to use the new table is done as follows.

1. Find the default table number for the desired sampling and bit rate by the **EPD** command. The number returned ranges between 120 and 239 and is the psychoacoustic table number used for the specified sampling and bit rate (called the *default table number*). Remember the second number returned by this command because it is usually used as the table number to store the modified table into (called the *suggested new table number*).
2. Execute the **EPL** command with the *default table number* to read the psychoacoustic table into memory and to download it to the encoder DSP.
3. Modify the psychoacoustic parameters with the **EPP** command until the desired audio quality is achieved.
4. Store the modified table in the *suggested new table number* by the **EPS** command.
5. Tell the encoder to use this new table for the specified sampling and bit rate by executing the **EPT** command.

It is possible and often useful to use the **EPT** command to assign multiple sampling and bit rates to the same table to minimize the table building errort.

7. Prima Logic Language

The conversion of events, such as silence detection, into actions, such as relay closures, is handled by the cdqPRIMA Logic Language (PLL). The PLL is a simple but powerful language designed specifically for the cdqPRIMA monitor and control.

At the cdqPRIMA, there are various inputs called events. Examples of events are switch closures and silence detection. These Events are all binary in nature and are on or off (high or low). Events are mapped into Actions by Boolean logic which includes AND, OR and NOT operators. The group of Events joined by the Boolean operators is called an Event Expression. The real time values of the various events can be displayed by executing the CEV command.

Approximately every .01 seconds, each Event Expression associated with and Action is evaluated and the corresponding Action set true or false.

Silence detection Events are generated by a silence detector.

The silence detector sensitivity is determined by various parameters. For example, the level and duration of silence must be defined which causes a silence Event. See the MQC, MQD, MQL and MQT commands.

The BER detector also has parameters which must be set. See the

MBD, MBL, MBR and MBU commands.

The Actions are binary also and thus are either true or false (high or low, on or off). This means that the output Action will do something such as open or close a relay, light or extinguish a LED. A real time snapshot of the Actions can be seen by executing the CRA command. The Actions are also latched. The latched values are read by the CLA command and are cleared by the CAR command. The purpose of the latched Actions concept is to see if an Action occurred anytime in the past. This allows the detection of transient Actions (Actions which occur and then disappear).

Actions can also be the result of transitions of an Event or Event Expression from low to high or high to low. For example, the LED display might display a message when the silence detector goes from not audio present (not silent) to no audio detected (silent).

Actions perform operations at the local cdqPRIMA, such as close a relay. Actions can also be exported to a far end cdqPRIMA and can be used as input Events at the far end. There are 12 logical connections from

the near end cdqPRIMA to a far end cdqPRIMA. These connections are called links. These links are numbered from 0 to 11.

The some of the Actions described above are physical. They actually do something which can be observed. There are two other classes of

Actions which are logical (non-physical). The first type has already been discussed and are the links between the near and far end cdqPRIMA's.

The second type of logical action is called a Virtual Action. These are also Boolean. When a Virtual Action is asserted, a cdqPRIMA Remote Control Command (PRCC) can be executed. Virtual Actions are executed when the Event Expression is high (asserted). Since Virtual Actions

are evaluated every .01 seconds, then the following PLL statement produces an unexpected result.

```
CEV VA0 CI0
```

As long as CI0 is high, then once every .01 seconds, Virtual Action 0 is executed. What is probable meant by this expression is that when CI0 changes from a low to a high, then Virtual Action 0 should be executed. In a discussion to follow, it will be seen that this result can be easily achieved.

Actions are named below. See Fig. 8 for reference.

RL0 = relay 0 contact closure
RL1 = relay 1 contact closure
RL2 = relay 2 contact closure
RL3 = relay 3 contact closure
RL4 = relay 4 contact closure
RL5 = relay 5 contact closure
RL6 = relay 6 contact closure
RL7 = relay 7 contact closure
SC1 = send cue LED
RC1 = receive cue LED
RLS = summary alarm relay
VA0 = virtual action 0
VA1 = virtual action 1
VA2 = virtual action 2
VA3 = virtual action 3
LN0 = action exported to far end cdqPRIMA on link 0
LN1 = action exported to far end cdqPRIMA on link 1
LN2 = action exported to far end cdqPRIMA on link 2
LN3 = action exported to far end cdqPRIMA on link 3
LN4 = action exported to far end cdqPRIMA on link 4
LN5 = action exported to far end cdqPRIMA on link 5
LN6 = action exported to far end cdqPRIMA on link 6
LN7 = action exported to far end cdqPRIMA on link 7
LN8 = action exported to far end cdqPRIMA on link 8
LN9 = action exported to far end cdqPRIMA on link 9
LN10 = action exported to far end cdqPRIMA on link 10
LN11 = action exported to far end cdqPRIMA on link 11
ESM = encoder summary alarm
DSM = decoder summary alarm

LN0..LN11 are exported to the far end cdqPRIMA while the other actions are performed only locally.

The exported Actions a sent to the far end whenever the Action changes state or whenever a link timer has expired. This link timer interval is set by the ELU command. The result is that the exported actions are repeatedly sent to the far end even if no change has occurred. This is an attempt to communicate with the far end even in the presence of bit errors on the transmission line.

Events are named as follows.

OI0 = optical isolator input 0
OI1 = optical isolator input 1
OI2 = optical isolator input 2
OI3 = optical isolator input 3
OI4 = optical isolator input 4
OI5 = optical isolator input 5
OI6 = optical isolator input 6
OI7 = optical isolator input 7
BER = decoder bit error detector
OOF = decoder out of frame detector
SEL = enc left channel silence detector
SER = enc right channel silence detector
SDL = dec left channel silence detector
SDR = dec right channel silence detector
S'. = encoder stereo silence detector
SD = decoder stereo silence detector
CI0 = computer input 0
CI1 = computer input 1
CI2 = computer input 2
CI3 = computer input 3
CI4 = computer input 4
CI5 = computer input 5
CI6 = computer input 6
CI7 = computer input 7
DDAPLL = decoder digital audio pll
EDAPLL = encoder digital audio pll
TI0 = timer 0 running
TI1 = timer 1 running
TS0 = timer 0 just expired
TS1 = timer 1 just expired
EPL = encoder pll locked
DPL = decoder pll locked
LN0 = imported action from far end cdqPRIMA on link 0
LN1 = imported action from far end cdqPRIMA on link 1
LN2 = imported action from far end cdqPRIMA on link 2
LN3 = imported action from far end cdqPRIMA on link 3
LN4 = imported action from far end cdqPRIMA on link 4
LN5 = imported action from far end cdqPRIMA on link 5
LN6 = imported action from far end cdqPRIMA on link 6
LN7 = imported action from far end cdqPRIMA on link 7
LN8 = imported action from far end cdqPRIMA on link 8
LN9 = imported action from far end cdqPRIMA on link 9
LN10 = imported action from far end cdqPRIMA on link 10
LN11 = imported action from far end cdqPRIMA on link 11
DSPD = decoder dsp dead
DSPE = encoder dsp dead

DSPR = reed-soloman dsp dead
DSPV = vu meter dsp dead

The events LN0 .. LN11 come from the far end cdqPRIMA.

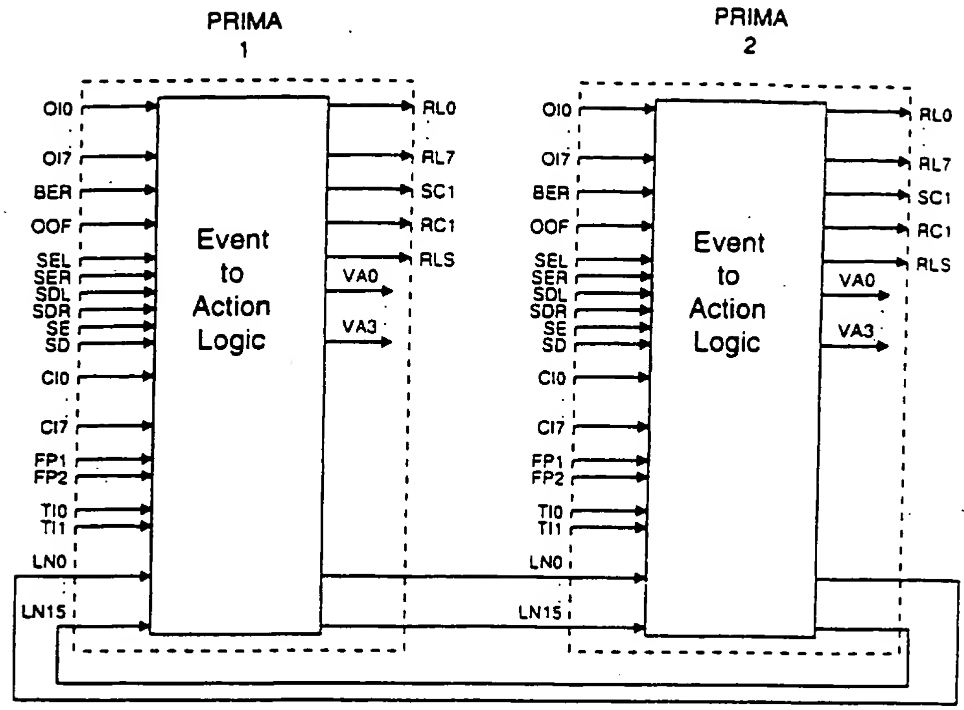
Depressing the front panel on and off cue buttons set and clear Event CI1 and CI2. Front panel cue button 1 corresponds to Event CI1 while button 2 corresponds to Event CI2

The front panel LED's labeled

SCUE1	(SC1)
RCUE1	(RC1)
ESUM	(ESM)
DSUM	(DSM)

are actually defined by the PLL. The are illuminated based on the Action shown in parenthesis in the above list. This means that the state of these LED's plus all the relays are completely user definable and can be remapped to met the needs of different applications.

The figure below shows the complete interconnection of two cdqPRIMA's with all the events and actions.



Events

OI0..OI7 - Optical Isolated / TTL inputs.

BER - Bit error rate detector

OOF - Out of frame detector

SEL - Encoder left channel silence detector
SER - Encoder right channel silence detector
SDL - Decoder left channel silence detector
SDR - Decoder right channel silence detector
SE - Encoder stereo silence detector
SD - Decoder stereo silence detector

CI0..CI7 - Computer simulated switch closures

FP1..FP2 - Front panel cue push buttons 1 and 2

TI0..TI1 - Timers 0 and 1

LN0..LN15 - Links from far end PRIMA

Actions

RL0..RL7 - Relay closures

SC1 - Send cue LED

RC1 - Receive cue LED

RLS - Summary relay closure

VA0..VA3 - Virtual actions

LN0..LN15 - Link to far end PRIMA

Figure 8
PRIMA Monitor and Control Block Diagram

The logical operators are

& = and

= or

! = not

The operator precedence is

! = 3

& = 2

= 1

where 3 is the highest precence.

An expression can have a maximum of 4 OR terms. If more than 4 OR terms are needed, a simple technique called DeMorgan's Theorm can be used to change OR's to AND's. There is no limit on AND terms.

DeMorgan's Theorm states

A = B & C

is equivalent to

```
A = !( !B # !C)
```

Thus the equation which contains many or terms

```
RLS = CI0 # CI1 # CI2 # CI3 # CI4 # CI5
```

can be rewritten as

```
RLS = !( !CI0 & !CI1 & !CI2 & !CI3 & !CI4 & !CI5 )
```

using DeMorgan's Theorm.

The + and - introduce the concept of actions base on transitions (edges). For example, if a LED message should occur when the front panel CUE 1 ON button is pressed, then the following PLL commands can be used.

```
CVA 0 CLM 10 HELLO WORLD
CEA VA0 +(CI1)
```

The first statement sets Virtual Action 0 to execute the CLM 10 HELLO WORLD command. The second line states that when computer input 1 (the CUE 1 button) changes from low to high (a + edge), then Virtual Action 0 si set to a 1 (executed).

The LED message can be immediately supressed when the CUE 1 OFF button is depressed if the following additional PLL statements are executed.

```
CVA 1 CLM 0
CEA VA1 -(CI1)
```

The first PLL statements associates the CLM 0 command with Virtual Action 1. The second statement activates VA1 on the high to low transition

(- edge) of the CUE 1 button.

The current PLL only allows parentheses around an entire expression.

The Actions and Events are considered identifiers.

The full use of the (and) operators will be allowed in future releases. See the command CDF for the default settings of the CEA commands.

Note that RI4 .. RI7 are not available on the 1xx series CODEC's but they are allowed in the PLL even though they won't do anything.

The ! operator can used in conjunction with the + and - operator.

For example

RL0 = OI0 Relay 0 follows the level of optical input 0

RL0 = !OI0 Relay 0 follows the inverted level of optical input 0

RL0 = +OI0 Relay 0 closes when OI0 changes from a low to a high

(note that there is no way to open relay 0)

RL0 = -OI0 Relay 0 closes when OI0 changes from a high to a low

(note that there is no way to open relay 0)

RL0 = +!OI0 Relay 0 closes when OI0 changes from a low to a high

(note that there is no way to open relay 0)

Look at another simple example. Optically isolated input 2 is connected to link 5 by the command

```
CEA LN5 OI2
```

This means that when optically isolated input 2 becomes a 1, then link 5 becomes a 1 and when OI2 becomes a 0, then link 5 is 0. Remember that the Link 5 Action is exported to the far end cdqPRIMA and becomes an input Event. More on this later. For now, we will just concentrate on the language.

394

The inverted input 2 is connected to link 5 by the command

```
CEA LN5 !OI2
```

This means that when optically isolated input 2 becomes a 0, then link 5 becomes a 1 and when OI2 becomes a 1, then link 5 is 0.

The next example is slightly more complicated.

```
CEA LN7 !(OI0 # CI1)
```

This states that link 7 will be low when either OI0 is high or CI1 is high.

The real time state of the encoder stereo silence detector state can be displayed by using the received cue LED on the front panel. This is accomplished by the following command.

```
CEA RC1 SE
```

The above command is useful when trying to debug the correct settings for the silence detector. A similar trick can be used for the BER detector.

An interesting example of the power of the PLL is shown below. The audio output can be muted when the BER detector raises to a high level and decoder is framed.

```
CVA 0 DMU BOTH

CEA VA0 +BER

CVA 1 DMU NORM

CEA VA1 -BER
```

The first command attaches the DMU BOTH command to Virtual Action 0. Virtual Action is executed when the BER detector transitions from low to high. The third line attaches the "restore DA output to normal" command to Virtual Action 1. The fourth line states that when the BER detector goes from a high BER count to a low BER count, then Virtual Action 1 is executed.

To reset the received cue led back to its origional definition, type

```
CEA RC1 LN8
```

A further PLL example is shown below.

```
CEA LN10 OI2 & SD # !CI3
```

In this example, link 10 is set high if EITHER OI2 and SD are high or if CI3 is low. Remember that & is higher precedence that # so the above expression could be written (if they were allowed) as follows.

```
CEA LN10 (OI2 & SD) # !CI3
```

395

Parenthesis are currently not allowed except around the entire expression. The above example would be much clearer if parenthesis were allowed.

An example of a Virtual Action is shown below

```
CVA 0 SD 4
CEA VA0 +SE
```

In this example, the CVA command assigns the operation of speed dialing entry 4 to the virtual action 0. The CEA command states that when the stereo encoder silence detector detects silence, then it sets virtual action 0 high. As just defined, the virtual action 0 would then perform the speed dial.

A last example is

```
CEA LN11 OI3 & OI4 & SD # CI3 & CI4 & SD # BER
```

It is left to the reader to decipher this command.

8. LCD Menu Tree Summary

The cdqPRIMA is controlled by use of the front panel keypad and the LCD display. Front panel control of the cdqPRIMA is accomplished by navigating a menu tree utilizing the MENU keys. The commands are organized into 4 main categories and these are

- Common commands
- Encoder commands
- Decoder commands
- Maintenance commands

The current position on the menu tree surrounded by square ([]) brackets while the current value of the command is enclosed in parenthesis (()).

The table below lists the entire contents of the menu tree and a further description of the command can be found under the cdqPRIMA remote control commands which are enclosed in parenthesis. For example, more information about the LCD TA dial command can be found under the CDI remote control command.

8.1 Common

General
- Password ....CPW Set user's password
- Version ....CVN Print software version number
- Set defaults ....CDF Set default parameters

Level LED's
- Mode ....CVU Set level meter mode
- Message ....CLM Display LED message
- Intensity.................CLI Set LED display intensity

Head Phones
- HP input ....CHP Set headphone audio source
- Volume ....CHV Set headphone volume level of current device

TA
- Dial ....CDI Dial TA phone number
- Hangup..................CHU Hangup a line or lines.
  - **Conn Time**
    - LCD Dsply.....CDC Display TA digital interface connect time.
    - Dsply Time ....CCS Get TA digital interface connect time
    - Clear Time .....CCR Clear TA digital interface connect time
- Conn Time Rst......CCR Clear TA digital interface connect time

Auto Answer	CAA	Set TA auto answer mode
Auto ReCon	CAC	Set TA auto-reconnection state
Dial Time out	CTO	Set TA dialing timeout
Connect	CTC	Connect to TA control port
RC Protocol	CTP	Set TA remote control protocol usage
RC Echo	CTE	Set TA remote control command response echo

Speed Dial

Speed Dial	CSD	Speed dial a number
View dir		View speed dial directory
Edit dir		Edit speed dial directory entry
Add entry	CSE	Enter a number in the speed dial directory
Del entry	CDS	Delete a speed dial number
Clear all	CSC	Clear all speed dial entries

Digital I/F

Define I/F	CIF	Set digital data interface type
Lp Bk Br	CBR	Set loopback bit rate
TA		
TA SPID	CSI	Set SPID for a Terminal Adaptor
TA ID	CLD	Set ID for a Terminal Adaptor
TA SW TYPE	CSW	Set switch type
TA loopback	CLB	Set loopback on a digital data interface

Other

Sys loopback	CSL	Set system loopback
DTR/CON	CDT	Set state of the DTR/CON line

RP Rmt Ctl

Set ID	CID	Set RS485 remote control ID
Interface	CRI	Set remote control interface type
Port baud	CRB	Set remote control baud rate
Protocol	CPC	Set remote control protocol usage
Echo	CRE	Set rear panel remote control command response echo

FP Rmt Ctl

FP baud	CFB	Set set front panel remote control baud rate
FP protocol	CFP	Set remote control protocol usage
Echo	CFE	Set front panel remote control command response echo

Time Code

Display src	CTI	Set Time Code readout source

Display spd	CTS	Print Time Code speed
Display last	CTL	Print last Time Code received
On/Off	CTT	Enable/disable Time Code
Status		
RS Read	CRA	Print the realtime value of the action word
LS Clear	CAR	Clear action word latched value
LS Read	CLA	Print action word latched value
PLL		
Prt evnt	CEV	Print event inputs
Stop timer	CCT	Cancel timer
Program	CEA	Set event to action logic
Async Anc Data		
MUX Baud Rate	CMA	Set ancillary data rate for MUX
DSP Baud Rate	CDR	Set ancillary data rate for encoder and decoder DSP
Mux	CAN	Set ancillary data mode
Sync Anc Data		
Enc bit rate	ESB	Set encoder synchronous bit rate
Enc clk edge	ESC	Set encoder synchronous clock edge
Dec bit rate	DSB	Set decoder synchronous bit rate
Dec clk edge	DSC	Set decoder synchronous clock edge
Hot Key	CHK	Define hot key
Virtual Act	CVA	Define virtual action

8.2 Encoder

General

Bit rate	EBR	Set encoder bit rate
Algorithm	EAL	Set encoder algorithm
Algorithm mode	EAM	Set encoder algorithm mode
Line fmt	ELI	Set encoder digital lines format
Sample rate	ESR	Encoder sampling rate
Audio source	EAI	Set encoder audio input source
Analog bw	EAB	Set encoder analog bandwidth
Volume	EHV	Set encoder headphone volumn level
Timing	ETI	Encoder timing
ACE	ESP	Set encoder scale factor protection
Calibrate AD	EAD	Calibrate AD converter

ISO Header

Copyright	ECR	Set encoder copyright bit in header
Emphasis	EEP	Set encoder emphasis bit in header
Original	EOR	Set encoder original bit in header
Protection	EPR	Set encoder protection bit in header
Private	EPI	Set encoder private bit in header

Contacts

Set Switch	ESW	Set a simulated switch

Psycho

Set Parm	EPP	Set psychoacoustic parameter
Reset	EPB	Load all default psychoacoustic parameters
Tbl Num	EPD	Get default psychoacoustic parameter table number
Load Tbl	EPL	Load psychoacoustic parameter table from flash
Store Tbl	EPS	Store psychoacoustic parameter table in flash
Assign Tbl	EPT	Assign psychoacoustic parameter table

8.3 Decoder

General

Bit rate	DBR	Set decoder bit rate
Independent	DIN	Set decoder - encoder interaction
Output SR	DDO	Set digital output sampling rate.
Line fmt	DLI	Set decoder digital lines format
Timing	DES	Enable decoder AES sync timing
Decoding mode	DCO	Set decoder decoding mode
ACE	DSP	Scale factor protection
Calibrate DA	DDA	Calibrate DA converter
Algorithm	DAL	Set decoder algorithm
Status bits	DRS	Display real time status

Audio out

Mute	DMU	Mute decoder output channels
Copy/Swap	DCS	Set channel copy/swap mode
Test tones	DMD	Set decoder maintenance diagnostic mode

8.4 Maintenance

Silence Det

Time left	MQC	Display quiet detector level time left
Set lvl	MQL	Set quiet detector level
Set time	MQT	Set quiet time duration
Read lvl	MQD	Display quiet detector level

Peak Det

Peak lvl	MPD	Display peak detector level

BER Det

Dsply Cnt	MBC	Display BER counter
Reset Cnt	MBR	Reset BER counter
Set Thresh	MBL	Set BER count rate limit
Up Cnt	MBU	Set BER up count rate
Down Cnt	MBD	Set BER down count rate

OOF Det

Dsply Cnt	MOC	Display OOF counter
Reset Cnt	MOR	Reset OOF counter
Set Thresh	MOL	Set OOF count rate limit
Up Cnt	MOU	Set OOF up count rate
Down Cnt	MOD	Set OOF down count rate

Graphic Tests

Graphics	MTM	Perform a test measurement

PRIMA Tests

En/Dis Tests	MET	Enable hardware tests
Hrdwre Tests	MHT	Perform hardware tests

Debug

Watch Port	MWP	Set watch port

Status

Version Num	MVN	Print software version number

Soft Dnld

Boot ROM	MBM	Boot the cdqPRIMA from ROM
FE Boot ROM	MRM	Boot the far end cdqPRIMA from ROM
RP RC Source	MRS	Set rear panel remote control uart source

BBM Sync	MSY	Sychronize RAM and BBM

402

9. cdqPRIMA Remote Control Commands

CAA Set TA auto answer mode

This command is used to set a digital interface TA to auto answer. It does this by asserting the DTR line on the Terminal Adaptor (TA). This command can also be used to hangup a connected call. If auto answer is set to NO, then a connected call (if any) is disconnected.

Once a digital interface line is set to no auto-answer, then it will not receive any calls. If a call is pending, and auto-answer is enabled, the the call will be answered.

See the CAD, CCR, CCS, CDI, CHU, CLD, CSI, CTC and CTO commands

CAA di ?	print auto answer status for digital interface **di**
CAA di aa	set auto answer status for digital interface **di** to **aa**

di = 1, 2, ... 6
aa = YES or NO

CAC Set TA auto-reconnection state

This command is used to set the TA auto-reconnection status. If **ad** is set to YES, then if a TA connection is dropped, it will automatically be re-established.

See the ?? commands.

CAC ?	print TA auto-reconnection state
CAC ad	set TA auto-reconnection state to **ad**

ad = YES or NO

CAN Set ancillary data mode

This command is used to set the ancillary data mux/demux configuration. See Fig 1 for a description of the various ancillary data configuration configurations.

See the CDR, DSB and ESB commands.

CAN ?	print current ancillary data configuration
CAN an	set ancillary data configuration to **an**

an = 0, 1, ... 6

CAR Clear action word latched value

This command clears the latched value of the action word.

403

The action outputs are printed as a hex number with the msb at the left and the lsb at the right. The meaning of the bits are as follows.

If a bit is high in the action word, then the corresponding action is also high.

BIT 0 RL0 - relay 0
BIT 1 RL1 - relay 1
BIT 2 RL2 - relay 2
BIT 3 RL3 - relay 3
BIT 4 RL4 - relay 4
BIT 5 RL5 - relay 5
BIT 6 RL6 - relay 6
BIT 7 RL7 - relay 7
BIT 8 SC1 - send cue 1 LED
BIT 9 RC1 - receive cue 1 LED
BIT 10 RLS - summary relay
BIT 11 VA0 - virtual action 0
BIT 12 VA1 - virtual action 1
BIT 13 VA2 - virtual action 2
BIT 14 VA3 - virtual action 3
BIT 15 unused
BIT 16 LN0 - link to far end PRIMA 0
BIT 17 LN1 - link to far end PRIMA 1
BIT 18 LN2 - link to far end PRIMA 2
BIT 19 LN3 - link to far end PRIMA 3
BIT 20 LN4 - link to far end PRIMA 4
BIT 21 LN5 - link to far end PRIMA 5
BIT 22 LN6 - link to far end PRIMA 6
BIT 23 LN7 - link to far end PRIMA 7
BIT 24 LN8 - link to far end PRIMA 8
BIT 25 LN9 - link to far end PRIMA 9
BIT 26 LN10 - link to far end PRIMA 10
BIT 27 LN11 - link to far end PRIMA 11
BIT 28 ESM - encoder summary alarm
BIT 29 DSM - decoder summary alarm
BIT 30 unused
BIT 31 unused

See the CEV, CEA, CLA, CRA, ELU and ESW commands.

CAR	clear the latched value of the action word

CBR Set loopback bit rate

This command is used to set the digital audio bit rate when the PRIMA is in loopback.

See the CLB and CSL commands.

CBR ?	print loop back bit rate
CBR lr	set loop back bit rate to **lr**

404

lr = 24, 32, 48, 56, 64, 96, 112, 128, 192, 224, 256, 384

CCR Clear TA digital interface connect time

This command is used to clear the number of seconds a terminal adaptor type of digital interface is connected.

The time connected can be read by the CCS command.

When the CCR command is executed, it clears the time in the timer.

See the CAA, CAD, CCS, CDI, CHU, CLD, CSI, CTC and CTO commands

CCR di clear the time in seconds a terminal adaptor is connected

di = 1, 2, ... 6

CCS Get TA digital interface connect time

This command is used to get the number of seconds a terminal adaptor type of digital interface is connected. When the TA enters the connect state, a timer for that digital interface is started and counts seconds. When the line is disconnected, the timer is stopped but not cleared.

The time line was connected is displayed by this command. If the digital interface TA is currently connected, this command will report the current elapsed connect time.

The timer can be set to 0 by issuing the CCR command.

This command is useful for monitoring the time a call is in progress.

See the CAA, CAD, CCR, CDI, CHU, CLD, CSI, CTC and CTO commands

CCS di print the time in seconds a terminal adaptor is connected

di = 1, 2, ... 6

CCT Cancel timer

This command is used to cancel an internal timer. This timer is used by the PRIMA Logic Language (PLL) to generate events which occur at some future time.

A timer cancelled by this command does not create any action.

See the CTM and CEA commands.

CCT tn cancel timer **tn**

tn = 0 or 1

405

CDC Display TA digital interface connect time.

This command is used to display the number of seconds a terminal adaptor type of digital interface is connected. When the display is requested, then it is displayed on the LCD screen. If may cover up part of another display.

When the TA enters the connect state, a timer for that digital interface is started and counts seconds is displayed. When the line is disconnected, the timer is stopped but not cleared and it is still displayed.

When the digital interface is again connected, the timer is reset and begins counting again.

There are two forms for the command.

CDC NO

CDC YES **di**

The first form is used to set to inhibit the display while the second form is used to display the connect time on digital interface **di**.

The CDC ? command has two possible responses. They are

NO

YES **di**

In the first case, no digital interface connect time is displayed. In the second case the connect time for DIF di is being displayed on the LCD display.

This command is useful for monitoring the time a call is in progress.

See the CAA, CAD, CCR, CDI, CHU, CLD, CSI, CTC and CTO commands

CDC ?	print the TA connect time display status
CDC NO	stop printing the connect time on lcd display
CDC YES di	display TA digital interface connect time on DIF **di**
	di = 1, 2, ... 6

CDF Set default parameters

This command is used to set the CODEC to the factory default values.

The default values are as follows:

CAA 1 YES	set auto answer on for line 1
CAA 2 YES	set auto answer on for line 2
CAA 3 YES	set auto answer on for line 3

406

CAA 4 YES	set auto answer on for line 4
CAA 5 YES	set auto answer on for line 5
CAA 6 YES	set auto answer on for line 6
CAN 2	set ancillary data port to configuration 2 (mux)
CBR 128	set loopback digital interface bit rate
CDC NO	don't display connect time
CDR 9600	set decoder dsp ancillary data rate
CEA LN0 !OI0	set default actions
CEA LN1 !OI1	
CEA LN2 !OI2	
CEA LN3 !OI3	
CEA LN4 !OI4	
CEA LN5 !OI5	
CEA LN6 !OI6	
CEA LN7 !OI7	
CEA LN8 CI1	
CEA LN9 CI2	
CEA LN10 BER	
CEA LN11 OOF	
CEA ESM !EPL	
CEA DSM !DPL # BER # OOF	
CEA RLS !EPL # !DPL # BER # OOF	
CEA RL0 LN0	
CEA RL1 LN1	
CEA RL2 LN2	
CEA RL3 LN3	
CEA RL4 LN4	
CEA RL5 LN5	
CEA RL6 LN6	
CEA RL7 LN7	
CEA SC1 CI1	
CEA RC1 LN8	
CEA VA0	
CEA VA1	
CEA VA2	
CEA VA3	
CFB 9600	set front panel remote control baud rate
CFP NO	set no front panel remote control protocol
CHK 1	set to nothing in the hot hey
CHK 2	set to nothing in the hot hey
CHK 3	set to nothing in the hot hey
CHK 4	set to nothing in the hot hey
CHK 5	set to nothing in the hot hey
CHK 6	set to nothing in the hot hey
CHK 7	set to nothing in the hot hey
CHK 8	set to nothing in the hot hey
CHP E	set headphone to encoder
CID 0	set to RS485 id 0
CIF 1 NONE	set to no digital interface
CIF 2 NONE	set to no digital interface
CIF 3 NONE	set to no digital interface
CIF 4 NONE	set to no digital interface
CIF 5 NONE	set to no digital interface
CIF 6 NONE	set to no digital interface
CLB 1 NORM	set no digital loopback on DIF 1

CLB 2 NORM	set no digital loopback on DIF 2
CLB 3 NORM	set no digital loopback on DIF 3
CLB 4 NORM	set no digital loopback on DIF 4
CLB 5 NORM	set no digital loopback on DIF 5
CLB 6 NORM	set no digital loopback on DIF 6
CLI STATUS 10	set status led intensity
CLI ENCODER 10	set encoder led intensity
CLI DECODER 10	set decoder led intensity
CMA 2400	set mux ancillary data baud rate
CPC NO	no protocol for remote communications
CRB 9600	set remote control baud rate
CRI 232	RS232 for remote communication
CSL NORM	no system loopback
CTC NONE	no connection to any TA
CTI NONE	no time code display
CTT OFF	no time code hardware
CTO 15	set TA dialing timeout in seconds
CVA 0	set virtual action 0 to empty
CVA 1	set virtual action 1 to empty
CVA 2	set virtual action 2 to empty
CVA 3	set virtual action 3 to empty
CVU LEVEL	set level meter to level (vu mode)
DAL MPEGL2	set to MPEG layer 2
DCO ISOCCS	set decoder decoding mode to ISO and CCS
DCS NONE	set decoder output to no copy or swap
DDO 48	set to 48 khz digital output
DES NOTREQ	set decoder sync timing not required
DHV 75	set decoder headphone volume
DIN NO	set decoder to operate together with encoder
DLI L1	set to no decoder line usage
DMD NORM	set decoder maintenance diagnostic mode to normal
DMU NONE	set decoder mute to none
DSB NONE	set no decoder synchronous ancillary data
DSP NO	set to no decoder scale factor protection
DTI NORMAUTO	set decoder timing to normal
EAL MPEGL2	set to MPEG layer 2
EBR 128	set encoder to 128k bit rate
ECR NO	set no copyright bit
EEP NO	set no emphasis bit
EHV 75	set encoder headphone volume
ELI L1	set to no encoder line usage
ELU 1	set to link messages every .1 sec
EOR NO	set no origional bit
EPI OFF	set no privicacy bit off
EPR YES	set protection bit
EPY 0 1	set psychoacoustic parameter type
EPY 1 2	set psychoacoustic parameter type
EPY 2 1	set psychoacoustic parameter type
EPY 3 2	set psychoacoustic parameter type
EPY 4 1	set psychoacoustic parameter type
EPY 5 3	set psychoacoustic parameter type
EPY 6 1	set psychoacoustic parameter type
EPY 7 1	set psychoacoustic parameter type
EPY 8 1	set psychoacoustic parameter type
EPY 9 3	set psychoacoustic parameter type

EPY 10 4	set psychoacoustic parameter type
EPY 11 3	set psychoacoustic parameter type
EPY 12 4	set psychoacoustic parameter type
EPY 13 3	set psychoacoustic parameter type
EPY 14 1	set psychoacoustic parameter type
EPY 15 3	set psychoacoustic parameter type
EPY 16 4	set psychoacoustic parameter type
EPY 17 3	set psychoacoustic parameter type
EPY 18 4	set psychoacoustic parameter type
EPY 19 4	set psychoacoustic parameter type
EPY 20 3	set psychoacoustic parameter type
EPY 21 3	set psychoacoustic parameter type
EPY 22 1	set psychoacoustic parameter type
EPY 23 1	set psychoacoustic parameter type
EPY 24 1	set psychoacoustic parameter type
EPY 25 1	set psychoacoustic parameter type
EPY 26 4	set psychoacoustic parameter type
EPY 27 3	set psychoacoustic parameter type
EPY 28 4	set psychoacoustic parameter type
EPY 29 4	set psychoacoustic parameter type
EPY 30 1	set psychoacoustic parameter type
EPY 31 1	set psychoacoustic parameter type
ESB NONE	set no encoder synchronous ancillary data
ESP NO	set to no encoder scale factor protection
ESR 48	set encoder sampling rate to 48
ESW CI0 OFF	set simulated switch 0 open
ESW CI1 OFF	set simulated switch 1 open
ESW CI2 OFF	set simulated switch 2 open
ESW CI3 OFF	set simulated switch 3 open
ESW CI4 OFF	set simulated switch 4 open
ESW CI5 OFF	set simulated switch 5 open
ESW CI6 OFF	set simulated switch 6 open
ESW CI7 OFF	set simulated switch 7 open
ETI NORM	set encoder timing to normal
MBD 1	set BER count down counter
MBL 1	set BER limit to 1
MBR	clear the BER counter
MBU 1	set BER count up counter
MOD 1	set OOF down counter
MOL 10	set OOF limit
MOU 2	set OOF up counter
MQL EL -60	set encoder left quiet threshold level
MQL ER -60	set encoder right quiet threshold level
MQL DL -60	set decoder left quiet threshold level
MQL DR -60	set decoder right quiet threshold level
MQL E -60	set encoder and decoder right quiet threshold level
MQL D -60	set encoder and decoder right quiet threshold level
MQT EL 10	set encoder left quiet threshold time
MQT ER 10	set encoder right quiet threshold time
MQT DL 10	set decoder left quiet threshold time
MQT DR 10	set decoder right quiet threshold time
MQT E 10	set encoder quiet threshold time
MQT D 10	set decoder quiet threshold time
MRS RP	set rear panel remote control source to rear panel
MWP NONE	set to no watch port

CDF sets the defaults into the cdqPRIMA

CDI Dial TA phone number

This command is used to dial a phone number on a specific digital interface. This command is used to set up a phone call and is primarily intended for testing. It can be used for setting up the lines individually.

To hangup a dialed line, use the CHU command.

See the CAA, CAD, CCR, CCS, CHU, CLD, CSI, CTC and CTO commands

CDI di db dn dial phone number dn on digital interface **di** at bit rate **db**

di = `1, 2, ... 6`

db = `56 or 64`

dn = 20 digit phone number

CDR Set ancillary data rate for encoder and decoder DSP

This command is used to set the ancillary data rate for the encoder and decoder DSP. The control processor are also involved in the ancillary data process. This data rate is used for communications from the mux to the encoder DSP and from the decoder DSP to the de-mux.

See the CAN, DSB and ESB commands.

CDR ? print encoder and decoder DSP ancillary data rate

CDR dr set encoder and decoder DSP ancillary data rate

dr = `300, 1200, 2400, 4800, 9600` and `38400`

CDS Delete a speed dial number

This command is used to delete a speed dial number.

See the CSC, CSD, CSE, CSF and CSN commands.

CDS sn delete speed dial number **sn**

sn = 0..255

CDT Set state of the DTR/CON line

410

This command is used to set the state of the DTR/CON line on non-TA type of interfaces.

See the CIF command.

CDT di ?	prints the state of the DTR/CON line on digital interface di		
CDT di st	set the state of DTR/CON line on digital interface **di** to st		
	di	=	1, 2, ... 6
	st	=	H or L

CEA Set event to action logic

This command is used to set the event to action logic. See section ?? for more details about this command.

See the CAR, CCT, CEV, CLA, CTM, CRA, CVA, ELU, ESW, MBD, MBL, MBR,MBU MQC, MQD, MQL and MPT commands.

CEA lf ?	print event to link connection for link ln		
CEA lf [el]	set event el to action **lf** connection		
	lf	=	LN0 .. LN11, RL0..RL7, SC1, RC1, RLS, VA0..VA3 (,), +, - and !'s
	el	=	optional event logic

CEV Print event inputs

This routine is used so print the compiled program for an action.

The event inputs are printed as a hex number with the msb at the left and the lsb at the right. The meaning of the bits are as follows.

If a bit is high in the event word, then the corresponding event is also high.

BIT 0	OI0 - optical isolator input 0
BIT 1	OI1 - optical isolator input 1
BIT 2	OI2 - optical isolator input 2
BIT 3	OI3 - optical isolator input 3
BIT 4	OI4 - optical isolator input 4
BIT 5	OI5 - optical isolator input 5
BIT 6	OI6 - optical isolator input 6
BIT 7	OI7 - optical isolator input 7
BIT 8	BER - decoder bit error detector
BIT 9	OOF - decoder out of frame detector
BIT 10	SEL - enc left channel silence detector
BIT 11	SER - enc right channel silence detector
BIT 12	SDL - dec left channel silence detector

411

BIT 13	SDR - dec right channel silence detector
BIT 14	SE - encoder stereo silence detector
BIT 15	SD - decoder stereo silence detector
BIT 16	CI0 - optical isolator input 0
BIT 17	CI1 - optical isolator input 1
BIT 18	CI2 - optical isolator input 2
BIT 19	CI3 - optical isolator input 3
BIT 20	CI4 - optical isolator input 4
BIT 21	CI5 - optical isolator input 5
BIT 22	CI6 - optical isolator input 6
BIT 23	CI7 - optical isolator input 7
BIT 24	DDAPLL - decoder digital audio pll locked
BIT 25	EDAPLL - encoder digital audio pll locked
BIT 26	TI0 - timer 0
BIT 27	TI1 - timer 1
BIT 28	TS0 - timer 0 stopped
BIT 29	TS1 - timer 1 stopped
BIT 30	EPL - encoder phase locked loop
BIT 31	DPL - decoder phase locked loop
BIT 32	LN0 - decoder link 0
BIT 33	LN1 - decoder link 1
BIT 34	LN2 - decoder link 2
BIT 35	LN3 - decoder link 3
BIT 36	LN4 - decoder link 4
BIT 37	LN5 - decoder link 5
BIT 38	LN6 - decoder link 6
BIT 39	LN7 - decoder link 7
BIT 40	LN8 - decoder link 8
BIT 41	LN9 - decoder link 9
BIT 42	LN10 - decoder link 10
BIT 43	LN11 - decoder link 11
BIT 44	DSPD - decoder dsp dead
BIT 45	DSPE - encoder dsp dead
BIT 46	DSPR - reed-soloman dsp dead
BIT 47	DSPV - vu dsp dead
BIT 48	FRAMED - decoder dsp framed

See the CEA, CAR, CLA, CRA, ELU and ESW commands.

CEV ev print event inputs ev

ev = ALL,

OI0..OI7, BER, OOF, SEL, SER,
SDL, SDR, SE, SD, CI0..CI7,
DDAPLL, EDAPLL,
TI0, TI1, TS0, TS1, LN0..LN11,
ESM, DSM, EPL,
DPL, DSPD, DSPE, DSPR, DSPV,
FRAMED

CFB Set set front panel remote control baud rate

This command is used to set the front panel remote control baud rate.

The baud rate can be 1200, 2400, 4800, 9600 or 38400 baud.

See the CFE and CFP commands.

CFB ?	print front panel remote control baud rate
CFB fb	set front panel remote control baud rate to **fb**
	fb = 1200, 2400, 4800, 9600 or 38400

CFE Set front panel remote control command response echo

This command is used to set the front panel remote control command echo. When downloading new software in flash, it is advisable to turn off command echo to speed the download process.

See the CFB and CFP commands.

CFE ?	print front panel remote control command response echo state
CFE re	set front panel remote control command response echo state to **re**
	re = YES or NO

CFP Set remote control protocol usage

This command forces the encoder to utilize a protocol on all front panel remote control messages. If no protocol is used, then point to point communications is assumed (a pc is connected to only 1 encoder). If protocol is used, then each CODEC device must have an id set by the CID command. The protocol can then select the specified device. Protocol communication can be used for point to point and point to multipoint communication.

If protocol was not enabled and it is enabled, the response will be in protocol mode (even though the input command was not in protocol mode) with a BSN of 0.

See the CFB and CFE commands.

CFP ?	print remote control protocol mode
CFP fp	set remote control protocol mode to **fp**
	fp = YES or NO

CHK Define hot key

This command is used to define a hot key. A hot key is front panel push button f1 to f8 which, when pushed, activates a command. For example

```
CHK 2 CSD 5
```

assigns the PRIMA Remote Control Command

```
CSD 5
```

to hot key 5 2.

CHK hk ?	print command associated with hot key hk		
CHK hk cm	attach command cm to hot key hk		
	hk	=	1 .. 8
	cm	=	any cdqPRIMA Remote Control Command (PRCC)

CHP Set headphone audio source

This command is used to set the headphone audio source.

The possibilities are the encoder (E, EL or ER), decoder (D, DL or DR) or mute (M).

For both the encoder and the decoder, the exist the possibilities of both channels(E or D), left channel only (EL or DL) or right channel only (ER or DR).

See the CHV, DHV and EHV commands.

CHP ?	print headphone audio source		
CHP hp	set headphone audio source to hp		
	hp	=	E, EL, ER, D, DL, DR or M

CHU Hangup a line or lines.

This command is used to hang up a connected line. It can only be used on digital interface lines designated as TA's.

The command to connect a line is CDI. To connect multiple lines, use the CAD command.

See the CAA, CAD, CCR, CCS, CDI, CLD, CSI, CTC and CTO commands

CHU df	hangup a line or lines.		
	df	=	ALL, 1, 2, ... 6

CHV Set headphone volume level of current device

414

This command is used to set the volume level of the currently selected device (encoder or decoder). The level applies to the selected device and controls the level of the audio output to the headphone jack.

See the CHP, DHV and EHV commands.

CHV ?	print headphone volume level of currently selected device		
CHV hv	set decoder headphone volume to **hv**		
	hv	=	0 .. 127, + or -

CID Set RS485 remote control ID

This command is used to set the RS485 id of the CODEC. This ID is used by remote control software to address the CODEC in an RS485 environment.

CID ?	prints the RS485 ID		
CID id	set the RS485 ID to id		
	id	=	0, 1, ... 30

CIF Set digital data interface type

This command is used to set the type of digital data interface. For the cdqPRIMA 2xx series, the interfaces are numbered from 1 through 6.

On the cdqPRIMA 1xx series, the interfaces are numbered 1 and 2.

If the interface is set to a TA, then auto answer is turned on.

If the interface type is X.21, V.35 or RS422, then the line state is set to CONNECTED (the CST command displays the connection status) permenently.

If the interface type is a type of TA, then the connection state is set to CONNECTED once the connection has been established.

See the CDT command.

CIF di ?	prints the interface type for digital interface di		
CIF di it	set digital interface di to it		
	di	=	1, 2, ... 6
	it	=	TA101, TA201, TA202, X.21, V.35, RS422 or NONE

CLA Print action word latched value

This command prints the latched value of the action word.

If a specific bit is requested, then the resulting value is 0 or 1. If all is specified, the the entire action word is printed in hex with the right most bit (LSB) corresponding to BIT 0.

The action outputs are printed as a hex number with the msb at the left and the lsb at the right. The meaning of the bits are as follows.

If a bit is high in the action word, then the corresponding action is also high.

BIT 0	LN0 - link to far end PRIMA 0
BIT 1	LN1 - link to far end PRIMA 1
BIT 2	LN2 - link to far end PRIMA 2
BIT 3	LN3 - link to far end PRIMA 3
BIT 4	LN4 - link to far end PRIMA 4
BIT 5	LN5 - link to far end PRIMA 5
BIT 6	LN6 - link to far end PRIMA 6
BIT 7	LN7 - link to far end PRIMA 7
BIT 8	LN8 - link to far end PRIMA 8
BIT 9	LN9 - link to far end PRIMA 9
BIT 10	LN10 - link to far end PRIMA 10
BIT 11	LN11 - link to far end PRIMA 11
BIT 12	ESUM - encoder summary alarm
BIT 13	DSUM - encoder summary alarm
BIT 14	unused
BIT 15	unused
BIT 16	RL0 - relay 0
BIT 17	RL1 - relay 1
BIT 18	RL2 - relay 2
BIT 19	RL3 - relay 3
BIT 20	RL4 - relay 4
BIT 21	RL5 - relay 5
BIT 22	RL6 - relay 6
BIT 23	RL7 - relay 7
BIT 24	SC1 - send cue 1 LED
BIT 25	RC1 - receive cue 1 LED
BIT 26	RLS - summary relay
BIT 27	VA0 - virtual action 0
BIT 28	VA1 - virtual action 1
BIT 29	VA2 - virtual action 2
BIT 30	VA3 - virtual action 3
BIT 31	not used

See the CEV, CEA, CAR, CRA, ELU and ESW commands.

CLA aw print latched value bit aw of the action word

aw = `ALL, LN0..LN11, ESM, DSM, RL0..RL7, SC1,RC1,RLS,VA0..VA3`

CLB Set loopback on a digital data interface

This command is used to set a loopback at the digital interface whose

number is given by ta.

For the cdqPRIMA, the interfaces are numbered from 1 through 6.

In the LB state, any data sent to the digital interface by the encoder is "looped back" to the decoder.

The CLB type of loopback is performed at the digital interface such as the TA. The CSL loopback is performed before the signals reach the digital interface.

See the CBR and CSL commands.

CLB di ?	prints the digital interface type		
CLB di lb	set loopback state **lb** on digital interface **di**		
	di	=	`1, 2, ... 6`
	lb	=	`LB or NORM`

CLD Set ID for a Terminal Adaptor

This command is used to set the ID for the digital interface.

The ID is only used for TA's in North America.

See the CAA, CAD, CCR, CCS, CDI, CHU, CSI, CTC and CTO commands

CLD di ?	prints the ID for digital interface **di**		
CLD di Id	set ID for digital interface **di** to **Id**		
	di	=	`1, 2, ... 6`
	Id	=	20 digit number

CLI Set LED display intensity

This command is used to set the intensity of the LED display.

The intensity can range from 0 to 15 where 15 is the brightest intensity.

This command

CLI gr ?	print current intensity		
CLI gr iy	set LED group **gr** to intensity **iy**		
	gr	=	`STATUS, ENCODER and DECODER`
	iy	=	`0..15`

417

CLM Display LED message

This command is used to display a message on the LED screen. It cancancel an existing message on the LED screen.

For example

```
CLM 2 5 MIN TO AIR
```

displays the message 5 MIN TO AIR for 2 seconds.

Another example

```
CLM 20 BER OCCURRED
```

displays the message BER OCCURRED for 20 seconds.

The command

```
CLM 0
```

terminates the display of any message on the LED and returns the display to vu mode.

CLM du [ms] displays message **ms** for duration **du**

	du	=	0 .. message duration in seconds
	ms	=	any ascii message up to 30 characters

CMA Set MUX ancillary data baud rate

This command is used to set the MUX asynchronous ancillary baud rate. The MUX ancillary data rate is diffent from the DSP ancillary data baud rate.

See the CDR command.

CMA ?	print MUX ancillary baud rate		
CMA ma	set mux ancillary data baud rate to **ma**		
	ma	=	300, 1200, 2400, 4800, 9600 or 19200

COM Comment command

This command takes an arbitrary number of arguments and ignores them. It is intended to be used for comments in a batch file.

COM x1 x2 x3 .. comment command (no operation)

x1 = any set of characters

x2 = any set of characters

CPC Set remote control protocol usage

This command forces the encoder to utilize a protocol on all remote control messages. If no protocol is used, then point to point communications is assumed (a pc is connected to only 1 encoder). If protocol is used, then each CODEC device must have an id set by the CID command. The protocol can then select the specified device. Protocol communication can be used for point to point and point to multipoint communication.

If protocol was not enabled and it is enabled, the response will be in protocol mode (even though the input command was not in protocol mode) with a BSN of 0.

If the RS-485 remote control interface is selected via the CRI command, then multiple CODEC's (up to 30) can be on the bus.

CPC ? print remote control protocol mode

CPC pc set remote control protocol mode to **pc**

pc = YES or NO

CPW Set user's password

This command allows the user to enter a password, thus raising his/her security level

CPW? prints the previous password

CPW pw enter the next password **pw**

pw = a big decimal number

CQQ Print command summary for common commands

This command is used to print a summary of all the Cxx commands.

See the DQQ, EQQ, MQQ and QQQ (HELP) commands.

CQQ print command summary

CRA Print the realtime value of the action word

419

BAD ORIGINAL

This command prints the realtime value of the action word.

If a specific bit is requested, then the resulting value is 0 or 1. If all is specified, the the entire action word is printed in hex with the right most bit corresponding to BIT 0.

See CLA for definition of the hex representation of the action word.

See the CEV, CEA, CAR, CRA, ELU and ESW commands.

CRA aw print realtime value bit **aw** of the action word

aw = ALL, LN0..LN11, RL0..RL7, SC1, RC1, RLS, VA0..VA3, ESM, DSM

CRB Set remote control baud rate

This command is used to set the remote control interface baud rate.

The baud rate can be 1200, 2400, 4800, 9600 or 38400 baud.

CRB ? print remote control baud rate

CRB rb set remote control baud rate to **rb**

rb = 1200, 2400, 4800, 9600 or 38400

CRE Set rear panel remote control command response echo

This command is used to set the rear panel remote control command echo. When downloading new software in flash, it is advisable to turn off command echo to speed the download process.

See the CRB command.

CRE ? print rear panel remote control command echo state

CRE re set rear panel remote control command echo state to re

re = YES or NO

CRI Set remote control interface type

This command is used to set the remote control interface type to RS232 or RS485.

CRI ? print remote control input source

CRI ri set remote control input source **ri**

ri = 232 or 485

CSC Clear all speed dial entries

This command is used to clear all speed dial entries. See the CDS, SD, CSE, CSF and CSN commands.

CSC clear all speed dial entries

CSD Speed dial a number

This command is used to speed dial a number.

See the CDS, CSC, CSE, CSF and CSN commands.

CSD sn ? prints the description for speed dial number **sn**

CSD sn speed dials speed dial number **sn**

sn = 3 digit speed dial number

CSE Enter a number in the speed dial directory

This command is used to insert a speed dial number in the directory

Not all combinations of entries are possible. The first decision is to determine the line format. This breaks down into two general catagories and these are H221 and not H221.

For the H221 case, then the

- bit rate (**br**) determines the number of connected lines
- sampling rate (**sr**) must be 32, 44 or 48
- encoder algorithm (**ea**) must be MPEGL2
- decoder algorithm (**da**) must be MPEGL2

The actual bit rate used will be determined by the number of lines connected. The lines called must utilize 64 kbs only. H.221 cannot currently handle n * 56 kbs.

For the L1 .. L6 case, then any of the rest of the parameters may be used. In this case, one phone number must be supplied. If more than one phone number is supplied, the the data is broadcast to each of the connected lines.

For CCSL12 .. CCSL56, then

- bit rate (**br**) must be set to 112 or 128
- sampling rate (sr) must be 32 or 48
- encoder algorithm (**ea**) must be MPEGL2, CCSO, CCSN

421

- decoder algorithm (**da**) must be MPEGL2, CCSO, CCSN
- two phone numbers (**d1** and **d2**) must be supplied

See the CDS, CSC, CSD, CSF and CSN commands.

CSE na ? print speed dial entry **na**

CSE na br sr ea em el NO d1 d2 d3 d4 d5 d6

CSE na br sr ea em el YES da dl li d1 d2 d3 d4 d5 d6

na	=	name of entry		
br	=	24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 192, 224, 256, 320, 384, A		
sr	=	16, 22, 24, 32, 44 or 48		
ea	=	da	=	MPEGL2, CCSO, CCSN or G.722
em	=	M, DM, JS, S		
el	=	**dl**	=	COMH221, L1, L2, L3, L4, L5, L6, CCSL12, CCSL13, CCSL14, CCSL15, CCSL16, CCSL23, CCSL24, CSL25, CCSL26, CCSL34, CCSL35, CCSL36, CCSL45, CCSL46, CCSL56
in	=	YES or NO		
d1	=	20 digit phone number		
d2	=	20 digit phone number		
d3	=	20 digit phone number		
d4	=	20 digit phone number		
d5	=	20 digit phone number		
d6	=	20 digit phone number		

CSF Print first of speed dial entry

422

description. If the optional parameter sh is set to A (abbreviated), only the entry number and speed dial description are displayed.

To print subsequent entries, see the CSN command.

This command prints the first entry in the speed dial list. The list is alphabetical by See the CDS, CSC, CSD, CSE and CSN commands.

CSF [sh]	print first speed dial entry		
	sh	=	A

CSI Set SPID for a Terminal Adaptor

This command is used to set the SPID for the digital interface. The SPID is only used for TA's in North America.

See the CAA, CAD, CCR, CCS, CDI, CHU, CLD, CTC and CTO commands

CSI di ?	prints the SPID for digital interface **di**		
CSI di sd	set SPID for digital interface **di** to **sd**		
	di	=	1, 2, ... 6
	sd	=	20 digit number

CSL Set system loopback

This command is used to set the system into loopback. Individual digital interfaces may be looped back (see the CLB command) but this command generates a loopback deeper inside the CODEC.

If the cdqPRIMA is powered down and powered up, the state of the loop back is NOT forgotten and the unit is set to to the state before the power was removed.

See the CBR and CLB commands.

CSL ?	print system loopback state		
CSL sl	set system loop back state to state sl		
	sl	=	NORM or LB

CSN Print next speed dial entry

This command prints the next entry in the speed dial list. The list is alphabetical by description. If the optional parameter sh is setto A (abbreviated), only the entry number and speed dial description are displayed.

See the CSF command for a description of the command output.

423

When the end of the list is reached, the message

END OF LIST

is displayed.

If the CSN command is given again after the END OF LIST is displayed, the first entry will be displayed. This means that continually entering the CSN command will repeatedly traverse the speed dial list.

See the CDS, CSC, CSD, CSE, and CSF commands.

CSN [sh]	Print next speed dial entry		
	sh	=	A

CST Report CODEC status

This command reports the general status of the CODEC.

CST report status

CSW Set switch type for a Terminal Adaptor

This command is used to set the switch type for the TA

type of digital interface. The switch type refers to the telephone company central office switch. Switches are made by such companies such as AT&T, Northern Telecom, Seimens .... These switches run different versions of ISDN software. This command sets the TA to work with a particular type of switch software.

The digital interface, di, can be either of the interfaces for the port. For example. if the TA is in the DIF23 slot, then di can be either 2 or three to set the switch type.

See the CAA, CAD, CCR, CCS, CDI, CHU, CLD, CTC and CTO commands

CSW di ?	prints the switch type for digital interface **di**		
CSW di si	set switch type for digital interface **di** to si		
	di	=	1, 2, ... 6
	North America		
	si	=	NI1, 5E6, 5E8 or NTI

CTC Connect to TA control port

424

This command is used to connect to the TA control port. This allows access to all the TA functionality. In partiºular, it allows configuration and call monitoring.

The TA control ports are named as follows

DIF12	DIF1 and DIF2
tc	digital interface
DIF34	DIF3 and DIF4
DIF56	DIF5 and DIF6

If tc is set to DIF1, then communication is established via DIF1 to a far end cdqPRIMA. The far end cdqPRIMA must be in the ISDN communication mode. This can be done at the far end by executing the MFC command or by sending the in-band MFC command to the far end (>>MFC)

See the CAA, CAD, CCR, CCS, CDI, CHU, CLD, CSI and CTO commands

CTC ? print current TA connection

CTC tc set connection to TA tc

tc = NONE, DIF12, DIF34 or DIF56

CTE Set TA remote control command response echo

This command is used to set the Terminal Adaptor (TA) remote control command echo. When downloading new software in flash, it is advisable to turn off command echo to speed the download process.

See the CTP commands.

CTE ? print TA remote control command response echo state

CTE re set TA remote control command response echo state to re

re = YES or NO

CTI Set Time Code readout source

This command is used to display the Time Code on the LCD display.

The displayed Time Code can be the Time Code input to the encoder, or the Time Code output from the decoder or no time code displayed.

If the timecode is displayed on the display, then depressing any front panel key or issuing the CTI NONE command terminates the Time Code display on the LCD.

This command is useful to check if time code is being received correctly by the encoder or the decoder.

See the CTL, CTS and CTT commands.

CTI ?	print Time Code readout source
CTI ti	set Time Code readout source to **ti**
	ti = NONE, INPUT, OUTPUT

CTL Print last Time Code received

This command is used to display the last time code received.

See the CTI, CTS and CTT commands.

CTL tf print last time code received for source **tf**

tf = INPUT or OUTPUT

CTM set timer timeout duration

This command is used to set an internal timer. This timer is used by the PRIMA Logic Language (PLL) to generate events.

This command starts the specified timer **tn** for the duration **ti**

The duration is in seconds.

See the CCT and CEA commands.

CTM tn ?	print timer **tn** time left in seconds
CTM tn tl	set timer **tn** to timeout in **tl** seconds
	tn = 0 or 1
	tl = 0..999999

CTO Set TA dialing timeout

This command is used to set the terminal adaptor dialing timeout. This timeout is used to terminate the dialing sequance for an individual TA.

See the CAA, CAD, CCR, CCS, CDI, CHU, CLD, CSI and CTC commands

CTO ? print TA dialing timeout value

426

CTO to set TA dialing timeout value **to** (in seconds)

to = 5..24

CTP Set TA remote control protocol usage

This command forces the control processor to use protocol protected communication on all TA remote control messages. If no protocol is used, then point to point communications is assumed (a pc is connected to only 1 encoder). If protocol is used, then each CODEC device must have an id set by the CID command. The protocol can then selectthe specified device. Protocol communication can be used for point to point and point to multipoint communication.

If protocol was not enabled and it is enabled, the response will be in protocol mode (even though the input command was not in protocol mode) with a BSN of 0.

See the CTE commands.

CTP ? print TA remote control protocol mode

CTP tp set TA remote control protocol mode to **tp**

tp = YES or NO

CTS Print Time Code speed

This command is used to display the Time Code speed. Time code can be 24, 25 or 30 frames per second.

See the CTI, CTL and CTT commands.

CTS tf print the time code speed for source **tf**

tf = INPUT or OUTPUT

CTT Enable/disable Time Code

This command is used to enable or disable the time code feature.

In the US, time code frames are transmitted at 30 frames per second. In Europe, they are transmitted at 24 frames per second.

The time code sub-system in the cdqPRIMA automatically senses andadapts to the input time code rate.

If time code is present at the encoder, the PRIMA attempts to deliver it to the far end decoder. To do this the ancillary data channel is used. This requires approximately 2400 bits per second of ancillary data.

If tt is set to OFF, then the time code input is always ignored and no ancillary data channel capacity is used.

If tt is set to ON and there is no time code signal present at the input, then no ancillary data resources are utilized.

See the CTI, CTL and CTS command.

CTT ? print current Time Code type

CTT tt set time code type to **tt**

tt = ON or OFF

CVA Define virtual action

This command is used to set a virtual action. The command associated with the virtual action is executed when the associated action becomes true. The Event - Action logic determines when an action becomes true.

For example

```
CVA 2 CSD 5
```

assigns the PRIMA Remote Control Command

```
CSD 5
```

to virtual action 2.

CVA va ? print command associated with virtual action va

CVA va cm define virtual action va as the command cm

va = 0 .. 3

cm = any PRIMA Remote Control Command (PRCC)

CVN Print software version number

This command is used to print the software version number for a thing in FLASH ram.

CVN tx print version and verify checksum of thing **tx**

tx = DSPD, DSPDX, DSPDXX, DSPV, DSPE, DSPEX, DSPR, CP or CPX

CVU Set level meter mode

This command selects the level meter mode. The level meter can be used to display the level of the audio as a normal vu meter. It can also be used to display the magnitude of the input as well as the position of the stereo image.

CVU ? print current level meter mode.

CVU vu set level meter mode to vu

vu = `LEVEL`, `IMAGE` or `PHASE`

DAL Set decoder algorithm

This command is used to set decoder algorithm.

See the DBR, DCO, DCS, DDA, DDO, DIN, DLI, DMD, DMU and DSP commands.

DAL ? print decoder algorithm

DAL al set decoder algorithm to **al**

al = `MPEGL2`, `CCSO`, `CCSN` or `G.722`

DBR Set decoder bit rate

This command is used to set the decoder digital audio bit rate.

Normally, the decoded bit stream dictates the sampling rate when inthe decoder operates independently from the encoder (see the DIN command). This command has no effect when DIN is set to NO.

Setting br to A lets the cdqPRIMA choose the bit rate based on the clock and the line type (see the DLI command).

See the DAL, DCO, DCS, DDA, DDO, DIN, DLI, DMD, DMU and DSP commands.

DBR ? print decoder bit rate

DBR br set decoder bit rate to br

br = `24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 192, 224, 256, 320, 384, A`

DCO Set decoder decoding mode

This command is used to control decoding of audio bit streams.

If ISO is selected, then only ISO layer 2 bit streams are decoded.

429

If ISOCCS is selected, then ISO layer 2 and older CCS bit streams are decoder. This command is for compatibility checking of bit streams.

See the DAL, DBR, DCS, DDA, DDO, DIN, DLI, DMD, DMU, DRS and DSP commands.

DCO ?	print decoder decoding mode		
DCO co	set decoder decoding mode		
	co	=	ISO or ISOCCS

DCS Set channel copy/swap mode

This command is used to control the audio output. It allows the left channel to be copied over the right channel (CLTOR), the right channel to overwrite the left channel (CRTOL) or the left and right channels to

be swapped (SWAP). If cs is set to NONE, then the output of the decoder is the same as received, ie. left channel to left channel and right channel to right channel.

This command is useful for controlling the action of the cdqPRIMA in the presence of mono audio signals.

See the DAL, DBR, DCO, DDA, DDO, DIN, DLI, DMD, DMU and DSP commands.

DCS ?	print decoder copy/swap mode		
DCS cs	set decoder copy/swap mode to cs		
	cs	=	NONE, CLTOR, CRTOL, SWAP

DDA Calibrate DA converter

This command is used to calibrate the DA converter. This operation takes about .1 second and during the calibration process, the audio output is muted. The DA converter is calibrated during power up but can be recalibrated at any time.

See the DAL, DBR, DCO, DCS, DDO, DIN, DLI, DMD, DMU and DSP commands.

DDA	calibrate da converter

DDO Set digital output sampling rate.

The digital audio is output from the MUSICAM decoder at the sampling rate specified in the MUSICAM header. This rate can then be converted to other rates via a sample rate converter.

The sample rate converter is capable of sampling rate changes between.51 and 1.99. For example, if the MUSICAM receiver received a bit streamwhich indicated that the sampling rate was 24 kHz, then the output sampling rate could be set to 32 or 44 kHz but not 48 kHz since 48 kHz would be a sampling rate conversion of 2.0 to 1. This is out of the range of the sampling rate converter.

The following table outlines the valid sampling rate conversions.

	Output Sampling Rates			
	29.5	32	44.1	48
Input Sampling Rates				
16	X			
22.05	X	X		
24	X	X	X	
32	X	X	X	X
44.1	X	X	X	X
48	X	X	X	X

Notice that the 16 kHz sampling rate cannot be output via the AES/EBU output port since it cannot be sample rate converted to any allowed value.

This command sets the digital audio (AES/EBU, SPDIF or optical) sampling rate. Setting do to M means that the sampling rate should follow the value contained in the MUSICAM audio frame.

See the DAL, DBR, DCO, DCS, DDA, DIN, DLI, DMD, DMU and DSP commands.

DDO ? print the decoder output sampling rate

DDO do set digital output sampling rate to **do**

do = 29, 32, 44 or 48

DES Enable decoder AES sync timing

431

BAD ORIGINAL

This command is used to enable/disable the use of the decoder AES/EBU sync signal. Normally, the AES/EBU sync signal for the decoder is used to determine the rate of the output of the AES/EBU decoder output. The AES/EBU decoder sync input can be ignored by setting es to DISABLE. See Fig. 7a, 7b and 7c for reference.

If there is no cable connected to the decoder sync input or DES is set to DISABLE, then the DA converter and the AES/EBU transmitter in the decoder is timed off the network clock. The exact value of the clock is phase locked to the network clock at a rate given by information in the received ISO/MPEG data stream.

If there is a sync signal present at the decoder sync input, then

the signal going to the decoder DA converter and to the AES/EBU transmitter is rate adapted to the frequency of the the received sync input.

DES ?	print status of decoder AES sync timing		
DES es	enable decoder AES sync timing		
	es	=	REQ or NOTREQ

DHV Set decoder headphone volumn level

This command is used to set the decoder volumn level. The level applies when the decoder is selected as the source of audio output to the headphone jack.

DHV ?	print decoder headphone volumn level		
DHV hv	set decoder headphone volumn to hv		
	hv	=	0 .. 127, + or -

DIN Set decoder - encoder interaction

This command is used to control the interaction between the decoder and the encoder. If in is set to NO, then the decoder and encoder interact. This is necessary for H.221 and one mode of two line CCS inverse multiplexing.

If ELI is set to COMH221, DIN is automatically set to NO.

Setting in to YES forces the decoder to operate completely indepently from the encoder. Any operation of the encoder has no effect on the decoder and visa versa.

See the DAL, DBR, DCO, DCS, DDA, DDO, DLI, DMD, DMU and DSP commands.

DIN ?	print decoder - encoder interaction

432

DIN in set decoder - encoder interaction to **in**

in = YES or NO

DLI Set decoder digital lines format

This command sets the format for the decoder digital interface lines.

This command is only valid if the decoder is set to operate independently. See the DIN command.

The **ls** parameter is defined as follows:

L1 indicates that only line 1 should be used.

L2 indicates that only line 2 should be used.

.

.

.

L6 indicates that only line 6 should be used.

CCSL12 .. CCSL56 indicates that CCS two line combined mode is to be used (see the ELI command).

See the DAL, DBR, DCO, DCS, DDA, DDO, DIN, DMD, DMU and DSP commands.

DLI ? print current decoder digital line format.

DLI ls set decoder digital line format ls

ls = L1, L2, L3, L4, L5, L6,
CCSL12, CCSL13, CCSL14,
CCSL15, CCSL16,
CCSL23, CCSL24, CCSL25,
CCSL26, CCSL34,
CCSL35, CCSL36, CCSL45,
CCSL46, CCSL56

DMD Set decoder maintenance diagnostic mode

This command is used to generate an output tone from the decoder. This is useful for setting levels in an analog system. It is also useful for checking the DA and digital outputs.

A 1000 and a 9600 Hz tone can be output to the left, right or both channels.

See the DAL, DBR, DCO, DCS, DDA, DDO, DIN, DLI, DMU and DSP commands.

If **md** is set to NORM, then the normal audio is output.

DMD ? print decoder maintenance diagnostic mode

DMD md set decoder maintenance diagnostic mode to **md**

md = NORM, 1KLEFT, 1KRIGHT, 1KBOTH, 10KLEFT, 10KRIGHT, 10KBOTH

DMU Mute decoder output channels

This command is used to mute the decoder audio output channels.

See the DAL, DBR, DCO, DCS, DDA, DDO, DIN, DLI, DMD and DSP commands.

DMU ? print the channels muted

DMU mu mute decoder outputs **mu**

mu = LEFT, RIGHT, BOTH or NONE

DQQ Print command summary for decoder commands

This command is used to print a summary of all the Dxx commands. See the CQQ, EQQ, MQQ and QQQ (HELP) commands.

DQQ print command summary

DRS Print decoder real-time status bits

This command is used to print the decoder status bits from the ISO/MPEG frame header. The emphasis, copyright, private, protection and copy bits are displayed by this command.

If the decoder is not framed, then the words

NOT FRAMED

are displayed.

If the decoder is framed, then the following is displayed.

ee o w v mm

The ee characters are one of the following

ee	*description*

NONE	no emphasis
50/15	50/15 microsecond emphasis
RES	reserved
J.17	CCITT J.17 emphasis

The o character is one of the following

o	*description*
O	origional version
C	copyed version

The w character is one of the following

w	*description*
W	copyrighted version
	non-copyrighted version

The v character is one of the following

v	*description*
V	the private bit is on
.	the private bit is off

The mm characters are one of the following

mm	*description*
PC	CRC algorithm is old ISO and frame type is CCS
PM	CRC is th old ISO and the frame type is ISO
MC	CRC is ISO and the frame type is CCS

435

MM	CRC is ISO and frame type is ISO
NC	there is no crc on the frame

See the DBR, DCO, DCS, DDA, DDO, DIN, DLI, DMD, DMU and DSP commands.

DRS ? print decoder real-time status bits

DRS print decoder real-time status bits

DSB Set decoder synchronous ancillary data bit rate

This command is used to set the decoder synchronous ancillary data bit rate. If the decoder is not independent, then the decoder synchronous ancillary data bit rate is set by the ESB command. If the decoder is independent, then the decoder synchronous ancillary data bit rate is set by this command.

See the CAN, CDR, DIN and ESB commands.

DSB ? print decoder synchronous ancillary data bit rate

DSB sb set decoder synchronous ancillary data bit rate to **sb**

sb = 8, 16, 32 or 64

DSP Scale factor protection

This command is used to enable or disable the use of scale factor protection. If scale factor protection checking is disabled, abit errors can have a much greater effect on the audio output than if scale factor protection is used.

If scale factor protection is used by the decoder, the encoder must also have scale factor protection enabled.

See the DAL, DBR, DCO, DCS, DDA, DDO, DIN, DLI, DMD and DMU commands.

DSP ? print decoder scale factor protection status

DSP sp set decoder scale factor protection to **sp**

sp = YES or NO

DTI Decoder timing

This command sets decoder timing source. See the Timing Section for a detailed description of this command.

436

DTI ?	print decoder timing source
DTI ts	set decoder timing source **ts**
	ts = NORMAUTO, INTAUTO, INT or AES

EAD Calibrate AD converter

This command is used to calibrate the AD converter. This operation takes about .1 second and during the calibration process, the audio output is muted.

The A-D converter is calibrated at power up. Calibrating the A-D converter before a critical recording results in the highest possible quality.

See the EAB, EAI, EAL, EAM, EBR, ELI, ESP and ESR commands.

EAD calibrate ad converter

EAI Set encoder audio input source

This command selects the type of input to the encoder. It can be either an analog or a digital input. The type of digital input (AES/EBU, SPDIF or optical) is selected by switches on the encoder.

See the EAB, EAD, EAL, EAM, EBR, ELI, ESP and ESR commands.

EAI ?	print current encoder audio source
EAI ai	set encoder audio source **ai**
	ai = A or D

EAL Set encoder algorithm

This command is used to set encoder algorithm. MPEGL2 set the encoder to output ISO/MPEG layer 2 frames. CCSN outputs CCS "new" frames. CCSO outputs CCS "old" frames. G.722 outputs the G.722 algorithm.

The various CCS algorithms are variations of the ISO/MPEG layer 2 standard. They were implemented before the standard was finalized and are included for backward compatibility with older CDQ200x CODEC's.

See the EAB, EAD, EAI, EAM, EBR, ELI, ESP and ESR commands.

EAL ?	print encoder algorithm
EAL al	set encoder algorithm to **al**
	al = MPEGL2, CCSO, CCSN or G.722

EAM Set encoder algorithm mode

This command is used to set encoder algorithm mode when the algorithm is MPEGL2, CCSO or CCSN. See the EAL command.

See the EAB, EAD, EAI, EAL, EBR, ELI, ESP and ESR commands.

EAM ?	print encoder algorithm mode		
EAM am	set encoder algorithm mode to **am**		
	am	=	S (stereo), JS (joint stereo), DM (dual mono) or M (mono)

EBR Set encoder bit rate

This command is used to set the encoder digital audio bit rate.

If A is selected, the bitrate is determined by the input digital interface clock and the line format (ELI).

If ELI is set to COMH221, the EBR is automatically to the necessary bitrate to match the number of connected lines. The bit rate set by this command is ignored in the COMH221 mode.

Upon changing to any line format, the bitrate will be set to the bitrate set by this command.

See the EAB, EAD, EAI, EAL, EAM, ELI, ESP and ESR commands.

EBR ?	print encoder bit rate		
EBR br	set encoder bit rate to **br**		
	br	=	24, 32, 40, 48, 56, 64, 80, 96,112, 128, 144, 160, 192, 224, 256, 320, 384, A

ECR Set encoder copyright bit in header

This command is used to enable or disable copyright bit in the ISOheader.

See the EEP, EOR and EPR commands.

ECR ?	print encoder copyright bit status		
ECR cr	set encoder copyright bit status to **cr**		
	cr	=	YES or NO

EEP Set encoder emphasis bit in header

This command is used to enable or disable emphasis bit in the ISO header.

See the ECR, EOR and EPR commands.

EEP ? print encoder emphasis bit status

EEP ep set encoder emphasis bit status to **ep**

ep = NO, 50, or J.17

EHV Set encoder headphone volumn level

This command is used to set the encoder volumn level. The level applies when the encoder is selected as the source of audio output to the headphone jack.

See the CHP, CHV and DHV commands.

EHV ? print encoder headphone volumn level

EHV hv set encoder headphone volumn to **hv**

hv = 0 .. 127, + or -

ELI Set encoder digital lines format

This command sets the format for the encoder digital interface lines.

The li parameter is defined as follows:

COMH221 indicates that multiple lines are combined utilizing H.221 L1 indicates that only line 1 should be used..

L2 indicates that only line 2 should be used.

.

.

.

L6 indicates that only line 6 should be used.

CCSL12 .. CCSL56 indicates that CCS two line combined mode is to be used.

If COMH221 is selected, usage of a maximum 6 lines is possible.

The actual number of lines is determined by the number of lines dialed. The encoder/decoder bit rate is set to 384 kbs and the decoder is set to not independent (DIN NO).

439

The TA lines are set CONNECTED when that are manually dialed, automatically dialed or connected to an incomming call.

See the EBR and DIN commands.

ELI ? print current encoder digital line format.

ELI li set encoder digital line format (li)

li = COMH221, L1, L2, L3, L4, L5, L6, CCSL12, CCSL13, CCSL14, CCSL15, CCSL16, CCSL23, CCSL24, CCSL25, CCSL26, CCSL34, CCSL35, CCSL36, CCSL45, CCSL46, CCSL56

ELU Set link message update rate

This command is used to set the link message update rate. The link messages are the exported part of the action word.

Link messages are sent to the far end cdqPRIMA every time the action word changes. If no changes in the action occur, the the link message is sent at a rate given by **ru**.

Ru of 1 means link message updates every .1 second, while ru = 5 means update link messages every .5 second.

An update rate of 0 turns off link messages.

See the CEV, CEA, CAR, CLA, CRA and ESW commands.

ELU ? print link message update rate

ELU ru set the link message update rate to **ru**

ru = 0 .. 10

EOR Set encoder original bit in header

This command is used to enable or disable original bit in the ISO header.

See the ECR, EEP and EPR commands.

EOR ? print encoder original bit status

EOR or set encoder original bit status to **or**

or = YES or NO

EPB Load all default psychoacoustic parameters

440

This command is used to load the default (factory supplied) psychoacoustic parameters. This is done by setting the tables for each sampling rate and bit rate to point to the factory supplied parameters.

This command is the same as executing the following two commands for each possible sampling rate and bit rate.

EPD **br sr**

to find the default table number for the sampling rate **sr** and bit rate **br**

EPT **tb br sr**

to set table tb (found from the above command) as the table to be used for sampling rate **sr** and bit rate **br**

See the EPD, EPL, EPP, EPS and EPT commands.

EPB	load all default psychoacoustic parameters

EPD Get default psychoacoustic parameter table number

This command is used to get the default pshchoacoustic parameter table number for the specified bit and sampling rates. The table number will be between 120 and 239. It also returns a second number to the right of the first number. This number is the suggested table number forthe user defined bit rate and sampling rate. This suggested table number can be ignored.

See the EPL, EPP, EPS, EPT and EPY commands.

EPD br sr get default psychoacoustic parameter table number

br = 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 192, 224, 256, 320, 384

sr = 16, 22, 24, 32, 44 or 48

EPI Set encoder private bit in header

This command is used to enable or disable private bit in the ISO header.

See the ECR, EEP, EOR and EPI commands.

EPI ?	print encoder private bit value
EPI pb	set encoder private bit value to **pb**

pb = ON or OFF

EPL Load psychoacoustic parameter table from flash

491

This command is used to load psychoacoustic parameters from FLASH into RAM memory. The these parameters become the current parameters and are downloaded to the encoder.

See the EPD, EPP, EPS, EPT and EPY commands.

EPL tb	load psychoacoustic parameter table from flash table tb
	tb = 0..239

EPP Set psychoacoustic parameter

This command is used to set a pshchoacoustic parameter.

The parameter type (EPY) must be set for each parameter before this command can be used.

See the EPD, EPL, EPS, EPT and EPY commands.

EPP pp ?	print the value of psychoacoustic parameter pp
EPP pp pv [0]	set psychoacoustic parameter pp to value pv with optional type **0** indicating **pv** is in hex
	pp = 0..31
	pv = floating point or integer number

EPR Set encoder protection bit in header

This command is used to enable or disable protection bit in the ISO header.

See the ECR, EEP, EOR and EPR commands.

EPR ?	print encoder protection bit status
EPR pr	set encoder protection bit status to **pr**
	pr = YES or NO

EPS Store psychoacoustic parameter table in flash

This command is used to store the current psychoacoustic parameters into flash memory.

Table numbers from 0 to 119 are the normal user tables. Table numbers from 120 to 239 are the default psychoacoustic tables and can only be overwritten by the system administrator.

See the EPD, EPL, EPP, EPT and EPY commands.

EPS tb	store psychoacoustic parameter table into flash table **tb**

tb = 0..119 for normal users

tb = 0..239 for system administrators

EPT Assign psychoacoustic parameter table

This command is used to assign a psychoacoustic parameter table to beused for a specified bit rate and sampling rate.

Psychoacoustic parameter tables are numbered from 0 to 239. Tables 0..119 hold user defined tables while tables 120..239 hold the system default tables.

If the EPT tb ? command is entered, pairs of numbers are returned. The left hand number is the bit rate an may be any value specified by

the br field. The right hand number is the sampling rate and may be any of the values specified in the sr field.

If no numbers are returned, then the specified table is not used by any sampling and bit rate.

If multiple pairs of numbers are returned, then the specified psychoacoustic table is used by more than one sampling / bit rate.

See the EPD, EPL, EPP, EPS and EPY commands.

EPT tb ?	print the bit rate and sampling rate for table tb
EPT tb br sr	assign psychoacoustic parameters table **tb** to be used for sampling rate sr and bit rate **br**

tb	=	0 .. 239
br	=	24, 32, 40, 48, 56, 64, 80, 96,112, 128, 144, 160, 192, 224, 256, 320, 384
sr	=	16, 22, 24, 32, 44 or 48

EPY Set psychoacoustic parameter type

This command is used to set the psychoacoustic paramter type. This command is used in conjunction with the EPP command.

See the EPD, EPL, EPP, EPS and EPT commands.

EPY pp ?	print psychoacoustic parameter type
EPY pp pt	set psychoacoustic parameter **pp** to type **py**

pp	=	0..31

443

pt = 0..4

EQQ Print command summary for encoder commands

This command is used to print a summary of all the Exx commands.

See the CQQ, DQQ, MQQ and QQQ (HELP) commands.

EQQ	print command summary

ESB Set encoder synchronous ancillary data bit rate

This command is used to set the encoder synchronous ancillary data bit rate. If the decoder is not independent, then the decoder synchronous ancillary data bit rate is also set to the same value.

See the CAN, CDR, DIN and DSB commands.

ESB ?	print encoder synchronous ancillary data bit rate
ESB sb	set encoder synchronous ancillary data bit rate to **sb**
	sb = 8, 16, 32 or 64

ESP Set encoder scale factor protection

This command is used to enable or disable the use of scale factor protection. If scale factor protection checking is disabled, a bit errors can have a much greater effect on the audio output than if scale factor protection is used.

If scale factor protection is used by the decoder, the encoder must also have scale factor protection enabled. Scale factor protection can be enabled in the encoder and not enabled by the decoder.

See the EAB, EAD, EAI, EAL, EAM, EBR, ELI and ESR commands.

ESP ?	print encoder scale factor protection status
ESP sp	set decoder scale factor protection to **sp**
	sp = YES or NO

ESR Encoder sampling rate

This command sets the sampling rate for the A-D converter or the digital audio input.

This only applies for the MPEGL2, CCSN and CCSO algorithms. For G.722 the sampling rate is fixed at 16 kHz.

See the EAB, EAD, EAI, EAL, EAM, EBR, ELI, ESP and ESR commands.

ESR ? print current encoder sampling rate

ESR sr set encoder sampling rate (**sr**) to one of the following:

sr = 16, 22, 24, 32, 44 or 48

ESW Set a simulated switch

This command is used to simulate a contact closure. This commandcauses actions based on the Event-Action logic.

See the CEV, CEA, CAR, CLA, CRA, and ELU commands.

ESW sw ? print status of simulated switch number **sw**

ESW sw ss set simulated switch number **sw** to state **ss**

sw = CI0 .. CI7

ss = ON or OFF

ETI Encoder timing

This command sets encoder timing source. The three choices are normal, internal crystal clock and aes/ebu.

ETI ? print encoder timing source

ETI te set encoder timing source ts

te = NORM, INT or AES

MBC Display BER counter

This command displays the BER counter. See the MBD, MBL, MBR and MBU commands. MBC ? Display the BER counter

MBC Display the BER counter

MBD Set BER down count rate

This command is used to set the BER down count rate. It is used in conjunction with the MBL command. For a detailed explaination of the MBD command, see the MBL command.

See the CEA, MBC, MBU, MBR and MBL commands.

MBD ? print current BER down count rate

MBD bd set BER down count rate to **bd**

bd = 0 .. 9

445

MBL Set BER count rate limit

This command is used to set the threshold limit for bit error rate.

If the bit error rate counter goes above this limit, then the BER event is set to true.

Each time a decoded frame is received (every 24 ms for 48k sampling rate MPEG I), the status of the BER bit is checked. The BER bit is set to a 1 by the decoder if MPEG frame protection is found and theframe CRC is in error. If the BER bit is on, the the BER counter is incremented by the value set by the MBU command. If the BER bit is off, then the BER counter is decremented by the value set by the MBD command. When the BER counter is equal or above the level set by the MBL command, then the BER event is set to true, otherwise it is set to false.

The contents of the BER counter may be displayed by the MBC command.

The BER counter may be set to 0 by the MBR command.

In a typical application of the BER commands, the following commands are used

`MBU 1`	set to count up by one on each frame with an error
`MBD 0`	set not to count down on ok frames
`MBR`	clear the counter
`MBL 1234`	wait until the ber count goes to 1234

The above sequence of commands can be used count the total number of bit errors and set the BER event when the count goes above 1234. The above sequence has the drawback that it never resets the BER count in the presence of good frames. The following remedies the situation by providing a leaky counter.

`MBU 10`	set to count up by one on each frame with an error
`MBD 1`	set not to count down on ok frames
`MBR`	clear the counter
`MBL 12340`	wait until the ber count goes to 12340

In the case above, every time a frame with a BER occurs, the count increments by 10. If a good frame occurs, then the count decrements by one. A long string of good frames erases a bad frame.

See the CEA, MBC, MBD, MBR and MBU commands.

MBL ?	print current BER up count rate
MBL bl	set BER up count rate to **bl**

446

bl = 0 .. 32767

MBM Boot the cdqPRIMA from ROM

This command is can only be used when the cdqPRIMA is executing out of the FLASH. It is used to boot the cdqPRIMA so that it runs out of ROM. In the ROM mode, all software can be downloaded including the control processor.

This command can be used to force the control processor into the software download mode. When control is passed to the ROM boot, the rear panel remote control port (RC) is connected to either the usual rear panel connector (RP) or digital interface port 1 (DIF1).

MBM rp boot the cdqPRIMA out of ROM and set RC port to **rp**

rp = RP or DIF1

MBO Boot the cdqPRIMA

This command is can only be used when the cdqPRIMA is executing out of the ROM. It is used to boot the cdqPRIMA so that it runs out of FLASH and has full functionality.

This command can be used after downloading new software into FLASH.

MBO boot the cdqPRIMA out of FLASH

MBR Reset BER counter

This command set the BER counter to 0.

See the CEA, MBC, MBD, MBL and MBU commands.

MBR Reset BER counter

MBU Set BER up count rate

This command is used to set the BER up count rate. It is used in conjunction with the MBL command. For a detailed explaination of the MBU command, see the MBL command.

See the CEA, MBC, MBD, MBR and MBL commands.

MBU ? print current BER up count rate

MBU bu set BER up count rate to **bu**

bu = 0 .. 9

MCP Set connect port

447

This command is used to connect the current remote port to a connect port. This allows a direct RS232 connection from the remote port the to connect port. This allows manual control of the connected port.

The remote port is the front panel or the rear panel remote control port.

MCP ? print current connect port

MCP cp set connect port to **cp**

cp = `NONE, 0 ... 7, TA0, TA1` or `TA2`

MET Enable hardware tests

This command is used to enable hardware tests. When hardware tests are enabled. then the normal operation of the the cdqPRIMA hardware is disabled. If hardware tests are enabled, then the various hardware subsystems may be tested.

See the MTM command.

MET et enable hardware tests

et = `ENABLE` or `DISABLE`

MHT Perform hardware tests

This command is used to perform hardware tests.

Setting ht to ALL performs all hardware tests See the MET and MTM command.

MHT ht perform hardware test **ht**

ht = `ALL, TC, IO, LED`

MOC Display OOF counter

This command displays the OOF counter.

See the MOD, MOL, MOR and MOU commands. MOC ? Display the OOF counter

MOC Display the OOF counter

MOD Set OOF down count rate

This command is used to set the OOF down count rate. It is used in conjunction with the MBL command. For a detailed explaination of the MOD command, see the MBL command.

See the MOC, MOU, MOR and MOL commands .

448

MOD ?	print current OOF down count rate
MOD od	set OOF down count rate to **od**
	od = 0 .. 9

MOL Set OOF count rate limit

This command is used to set the threshold limit for bit error rate.

If the bit error rate counter goes above this limit, then the OOF event is set to true.

Each time a decoded frame is received (every 24 ms for 48k sampling rate MPEG I), the status of the OOF bit is checked. The OOF bit is set to a 1 by the decoder if MPEG frame protection is found and theframe CRC is in error. If the OOF bit is on, the the OOF counter is incremented by the value set by the MOU command. If the OOF bit is off, then the OOF counter is decremented by the value set by the MOD command. When the OOF counter is above the level set by the MOL command, then the OOF event is set to true, otherwise it is set to false.

The contents of the OOF counter may be displayed by the MOC command.

The OOF counter may be set to 0 by the MOR command.

In a typical application of the OOF commands, the following commandsare used

`MOU 1`	set to count up by one on each frame with an error
`MOD 0`	set not to count down on ok frames
`MOR`	clear the counter
`MOL 1234`	wait until the ber count goes to 1234

The above sequence of commands can be used count the total number of bit errors and set the OOF event when the count goes above 1234.

The above sequence has the drawback that it never resets the OOF count in the presence of good frames. The following remedies the situation by providing a leaky counter.

`MOU 10`	set to count up by one on each frame with an error
`MOD 1`	set not to count down on ok frames
`MOR`	clear the counter
`MOL 12340`	wait until the ber count goes to 12340

449

In the case above, every time a frame with a OOF occurs, the count increments by 10. If a good frame occurs, then the count decrements by one. A long string of good frames erases a bad frame.

See the MOC, MOD, MOR and MOU commands.

MOL ?	print current OOF up count rate
MOL ol	set OOF count rate limit to **ol**
	ol = 0 .. 32767

MOR Reset OOF counter

This command set the OOF counter to 0.

See the MOC, MOD, MOL and MOU commands.

MOR Reset OOF counter

MOU Set OOF up count rate

This command is used to set the OOF up count rate. It is used in conjunction with the MOL command. For a detailed explaination of the MOU command, see the MOL command.

See the MOC, MOD, MOR and MOL commands.

MOU ?	print current OOF up count rate
MOU ou	set OOF up count rate to **ou**
	ou = 0 .. 9

MPD Display peak detector level

This command is used to read the level of the peak detector.

The value of the peak is measured in dB down from the maximum. For example a peak reading of -10 indicates that the highest peak value since the last peak level status request was -10 dB.

The largest value the peak can be is 0 dB.

Once the the peak value is read, it is set to -150 dB.

MPD pd	read peak detector level in dB down from maximum.
	pd = EL, ER, DL or DR

MQC Display quiet detector level time left

450

This command is used to read the the quiet detector time left counter. This is the time left in seconds before the specified input is declared as quiet.

If the time returned is between 0 and 255. If it is 255, then the quiet time has been set to 0 and the quiet detector for the input has been disabled.

See the CEA, MQD, MQL, MQD commands.

MQC qd	read quiet detector time left on input **qd**
	qd = EL, ER, DL, DL, E or D

MQD Display quiet detector level

This command is used to read the level of the quiet detector. This allows the monitoring of the average level of the audio signal averaged over 1 second.

The value reported is in dB down from the maximum value. Thus a value of-12 dB represents -12 dB down from the highest value.

The largest value returned is 0 dB.

The quiet detector detector level readings are updated approximately every 1 second. This means that if the MQD command is issued more often than once per second, it will return the same value.

A qd value of E means encoder left or encoder right, which ever has thehighest value. If the encoder left channel has a level has a quiet detector level of -87 dB and the encoder right channel has a value of -33 dB, the command MQD E would return a value of -33. A similar definition applies for the D command.

See the CEA, MQC, MQL and MQT commands.

MQD qd	read quiet detector level in dB down from maximum.
	qd = EL, ER, DL, DR, E or D

MQL Set quiet detector level

This command is used to set the threshold level for silence detection. The input audio level must be below this threshold for a certian period of time to be considered as a silent input.

The time duration is set by the MQT command.

See the CEA, MQD, MQT and MQC commands.

MQL qd ?	print quiet level for input **qd**
MQL qd ql	set the quiet level in dB relative to maximum input to **ql** for input **qd**

qd	=	EL, ER, DL, DR, E or D
ql	=	-1 to -120

MQQ Print command summary for decoder commands

This command is used to print a summary of all the Mxx commands.

See the CQQ, DQQ, EQQ and QQQ (HELP) commands.

MQQ	print command summary

MQT Set quiet time duration

This command is used to set the time in seconds that the input levelmust be below the threshold level before it is considered to be silent. The threshold level is set by the MQL command.

See the CEA, MQC, MQD and MQL commands.

MQT qd ?	print quiet time duration for input qd
MQT qd qt	set the quiet time duration to **qt** for input **qd**

qd	=	EL, ER, DL, DR, E or D
qt	=	0 (to set no quiet level checking on input qd) 1 .. 254 (number of seconds of quiet)

MRM Boot the far end cdqPRIMA from ROM

This command is used to force the far end cdqPRIMA to execute from boot ROM. This allows the farend to accept download information.

CAN must be set to mode 2 and the near and far end cdqPRIMA's must be operating in MPEGL2, CCSO or CCSN for proper operation.

See the CAN command.

MRM	boot the far end cdqPRIMA out of ROM

MRS Set rear panel remote control uart source

This command is used to set the source for the rear panel remote control UART. This UART may be connected to the rear panel connector or to DIF1. If the rear panel is selected, then the CRB command determines the baud rate for the port. If DIF1 is selected, then the clock on that DIF determines the bit rate. The later case is used for remote downloading of software via the DIF (ISDN).

See the MBM and MRM commands.

452

CRS ? print rear panel remote control UART source

CRS rp set rear panel remote control UART source to **rp**

rp = RP or DIF1

MSY Sychronize RAM and BBM

This command is used to write any unwritten bytes to nonvolitile memory. Many of the varibles that are kept in nonvolitile are first writen to standard RAM and at a later time, they are flushed to battery backed up RAM (BBM). This command forces all bytes which are in ram but not in BBM to be written.

This command can be issued just before turning off the power to insure that all "dirty" bytes are written to RAM.

MSY synchronize RAM and BBM.

MTM Perform a test measurement

This command is used to perform a test measurement.

See the MET command.

MTM ? print current test measurement in progress

MTM tm start test **tm**

tm = NONE, PHASEE, PHASED, FFTEL, FFTER, FFTDL or FFTDR

MVN Print software version number

This command is print the software version number of a thing.

See the ?? command.

MVN ty print software version number of thing ty

ty = ALL, CP, CPX, DSPD, DSPE, DSPV, DSPR DSPDX, DSPEX, DSPVX, DSPRX, DSPDXX, DSPEXX, DSPVXX, DSPRXX, TH0, TH1, TH2, TH3, TH4, TH5, TH6, TH7, TH8, TH9, TH10, TH11, TH12, TH13, TH14, TH15, TH16, TH17, TH18, TH19, TH20, TH21, TH22, TH23

MWP Set watch port

This command is used to set the output RS232 port for debugging messages from internal processes. For example, each time a relay or cue message is sent or received, then a message is output to the

port issuing the command. The watch port, when enabled, allows a look at internal communication in the cdqPRIMA.

MWP ? print current watch port

MWP wb set watch port to wp to watch **wb** items

wb = ABCDEFGHIJKLMNOPQRSTUVWXYZ or NONE

A = decoder HF2=0 dsp interrupts
B = encoder dsp interrupts
C = reed soloman dsp interrupts
D = vu dsp interrupts
E = event to action results (action word)
F = quiet detector scaled values from vu dsp
G = quiet detector raw values from vu dsp
I = decoder HF2=1 dsp interrupts
J = messages to TA port
K = messages from TA port
L = decoder DSP host vector messages
M = encoder DSP host vector messages
N = reed soloman DSP host vector messages
O = vu DSP host vector messages
P = phase check in phase process
Q = time code buffer to encoder
R = time code buffer from decoder
S = out going link word message
T = incomming link word message
U = peak detector scaled values from vu dsp
V = peak detector raw values from vu dsp
W = command from far end prima
X = response to far end prima
Y = command sent to far end prima
Z = response from far end prima

HELP Print command summary for all commands

This command is used to print a summary of all help commands.

See the CQQ, DQQ, EQQ and MQQ commands.

HELP print command summary

WHAT IS CLAIMED IS:

1. A system for compressing and decompressing data bit streams, said system being comprised of:

at least one external input means for receiving audio data;

a plurality of external input means for receiving ancillary data bit streams;

a multiplexer capable of receiving said plurality of ancillary data bit streams, said multiplexer producing a composite bit stream of ancillary data;

an encoder, containing a compression technique, which receives said audio data and said composite ancillary data bit stream and produces a resulting compressed data bit stream;

at least one digital interface output module for externally outputting said compressed data bit stream;

at least one digital interface input module for externally inputting an external compressed data bit stream;

a decoder, containing a decompression technique for decompressing data produced by said encoder compression technique, wherein said decoder receives said external compressed data bit stream and produces decompressed audio and composite ancillary data bit streams;

a demultiplexer capable of receiving said decompressed composite ancillary data bit stream, said demultiplexer producing a plurality of separate decompressed ancillary bit streams;

at least one external output means for outputting said decompressed audio data;

a plurality of external output means for outputting said decompressed ancillary data bit streams.

2. An audio CODEC for providing high quality digital audio comprising:

an analog to digital converter for converting an analog audio signal to a digital audio bit stream;

an encoder for compressing said digital audio bit stream;

a decoder for decompressing said compressed digital audio bit stream;

an output allowing a user to monitor the digital audio output; and

at least one control for allowing said user to adjust said digital audio output.

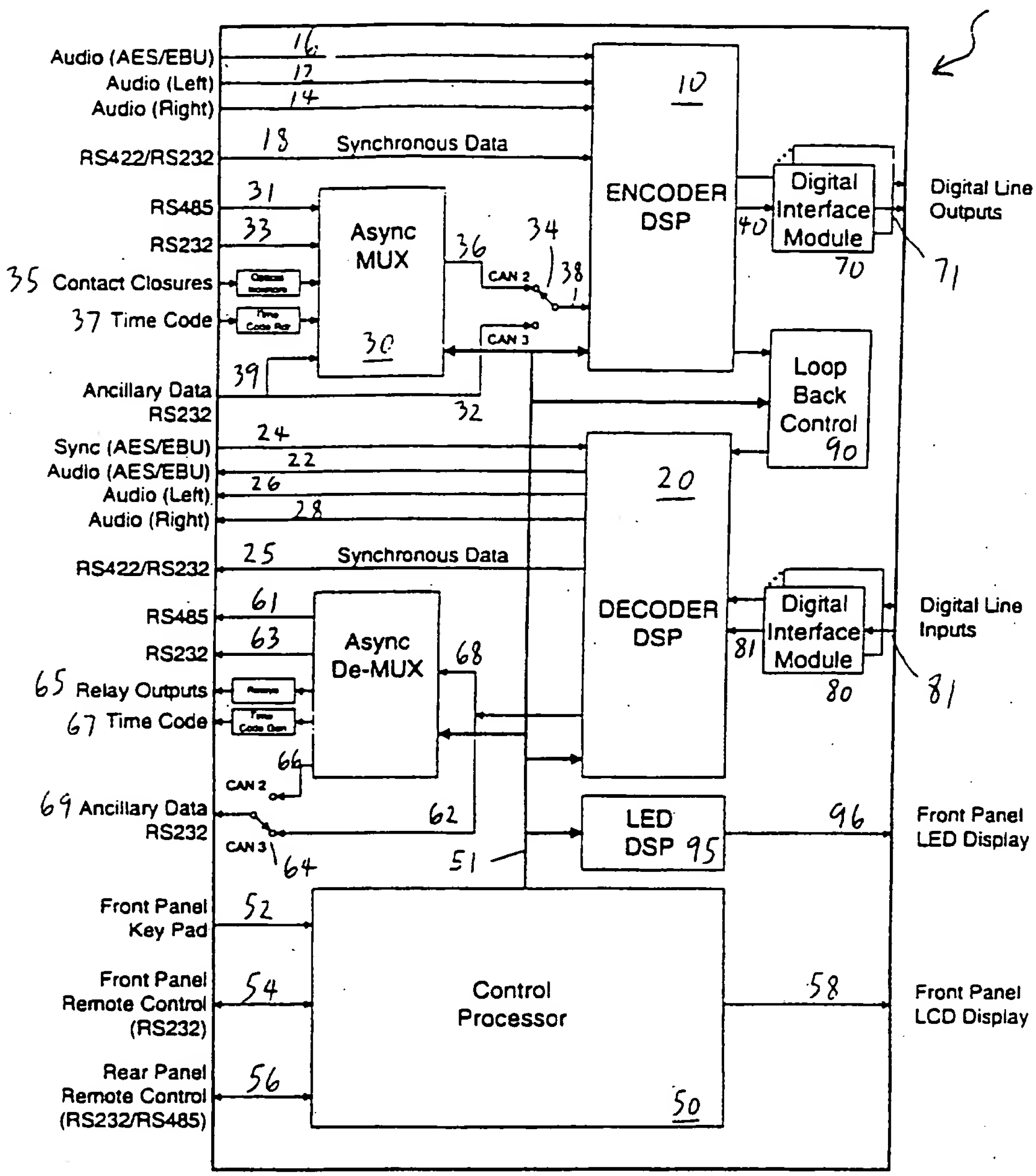
3. A method for providing high quality digital audio comprising the steps of:

providing an input analog audio signal;

providing at least one psycho-acoustic parameters;

converting said input analog audio signal into a digital signal;

coding said digital signal in accordance with said at least one psycho-acoustic parameter;



Audio (AES/EBU)
Audio (Left)
Audio (Right)
RS422/RS232
RS485
RS232
Contact Closures
Time Code
Ancillary Data
RS232
Sync (AES/EBU)
Audio (AES/EBU)
Audio (Left)
Audio (Right)
RS422/RS232
RS485
RS232
Relay Outputs
Time Code
Ancillary Data
RS232
Front Panel
Key Pad
Front Panel
Remote Control
(RS232)
Rear Panel
Remote Control
(RS232/RS485)
16
12
14
18
Synchronous Data
31
33
35
37
39
Async
MUX
30
36
34
38
CAN 2
CAN 3
32
ENCODER
DSP
10
40
Digital
Interface
Module
70
71
Digital Line
Outputs
Loop
Back
Control
90
24
22
26
28
25
Synchronous Data
61
63
65
67
Async
De-MUX
68
66
CAN 2
CAN 3
64
62
69
DECODER
DSP
20
81
Digital
Interface
Module
80
81
Digital Line
Inputs
LED
DSP
95
96
Front Panel
LED Display
51
52
54
56
Control
Processor
50
58
Front Panel
LCD Display

Figure 1

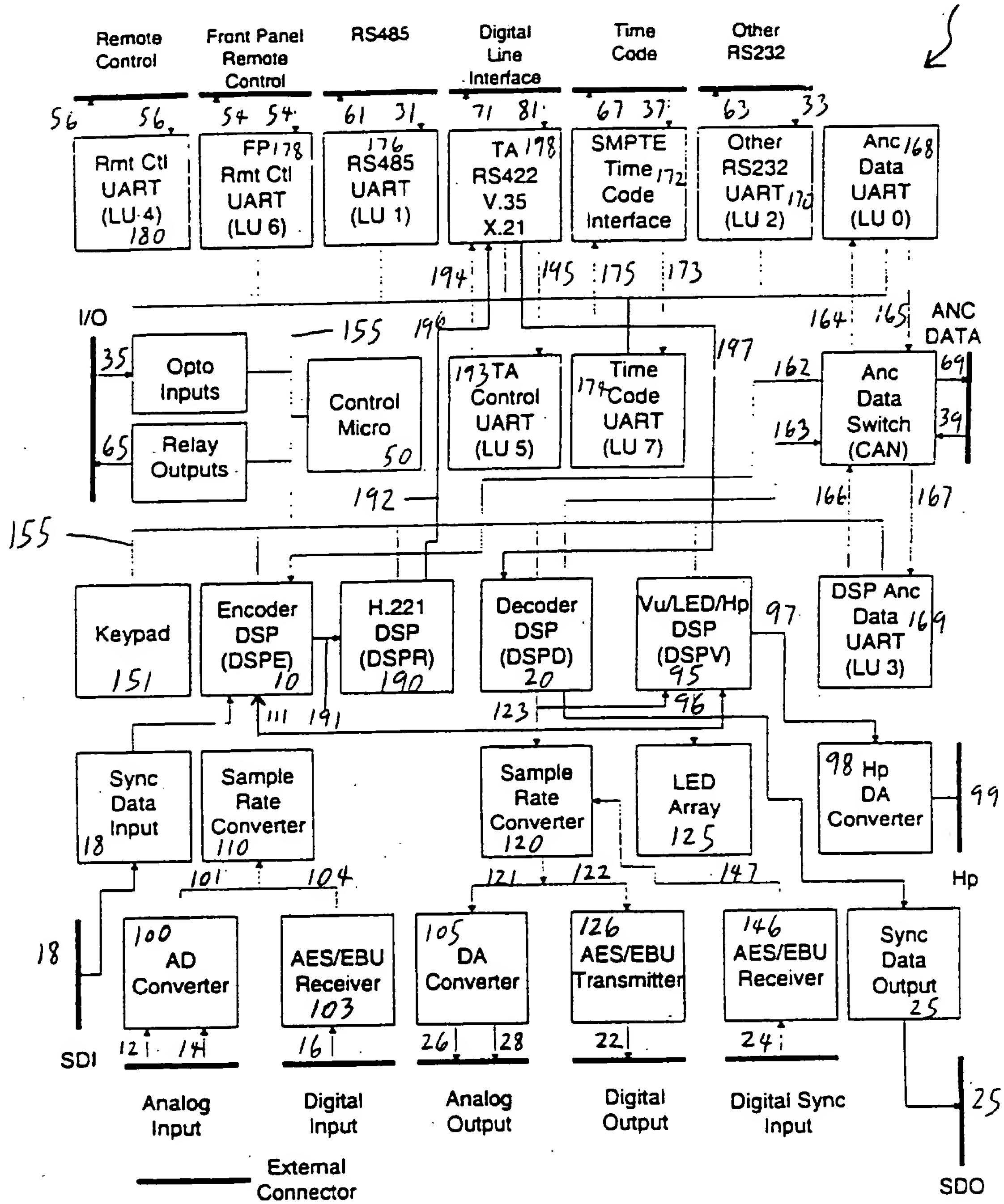

Remote Control
Front Panel Remote Control
RS485
Digital Line Interface
Time Code
Other RS232
Rmt Ctl UART (LU 4) 180
FP 178 Rmt Ctl UART (LU 6)
176 RS485 UART (LU 1)
TA 178 RS422 V.35 X.21
SMPTE Time Code Interface 172
Other RS232 UART (LU 2) 170
Anc Data UART (LU 0) 168
I/O
Opto Inputs
Relay Outputs
Control Micro 50
193 TA Control UART (LU 5)
174 Time Code UART (LU 7)
Anc Data Switch (CAN)
ANC DATA
Keypad 151
Encoder DSP (DSPE) 10
H.221 DSP (DSPR) 190
Decoder DSP (DSPD) 20
Vu/LED/Hp DSP (DSPV) 95
DSP Anc Data UART (LU 3) 169
Sync Data Input 18
Sample Rate Converter 110
Sample Rate Converter 120
LED Array 125
98 Hp DA Converter
Hp
100 AD Converter
AES/EBU Receiver 103
105 DA Converter
126 AES/EBU Transmitter
146 AES/EBU Receiver
Sync Data Output 25
SDI
SDO
Analog Input
Digital Input
Analog Output
Digital Output
Digital Sync Input
External Connector

Figure 2

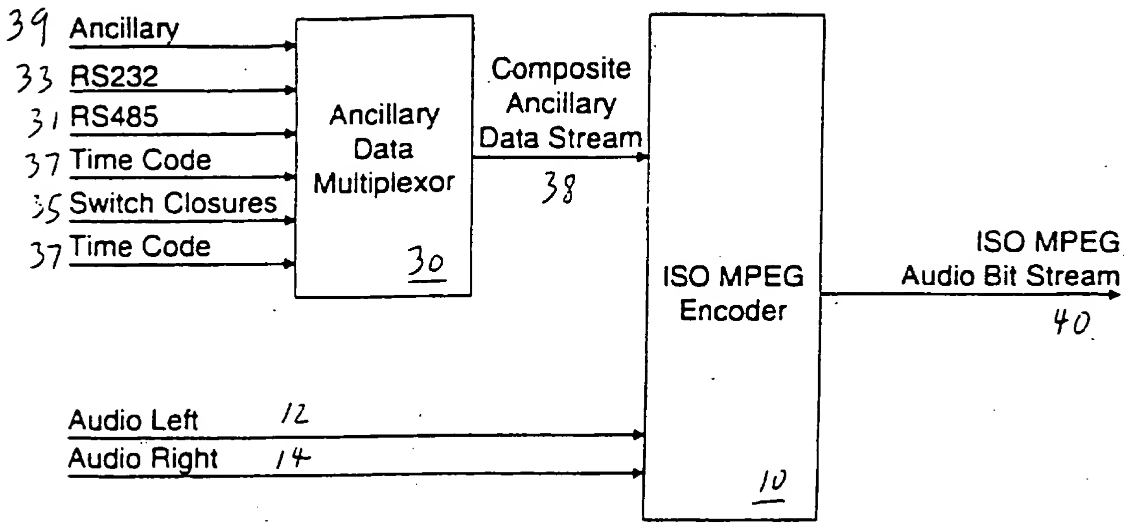


Figure 3

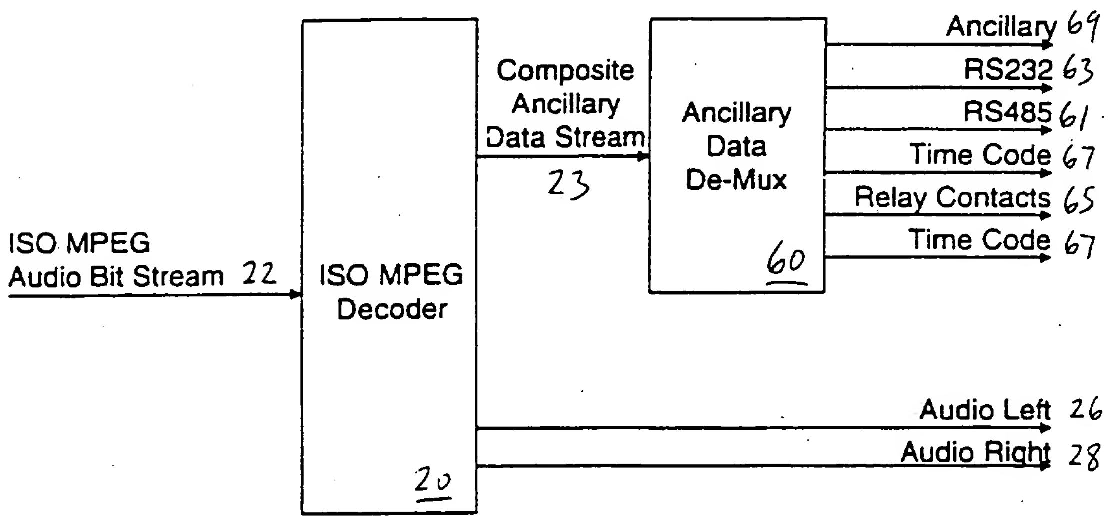


Figure 4

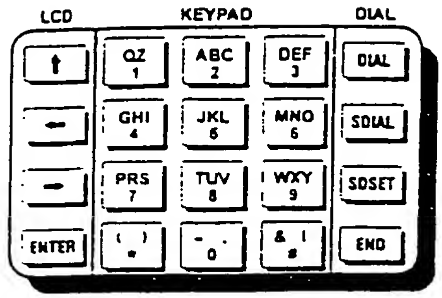


Figure 5

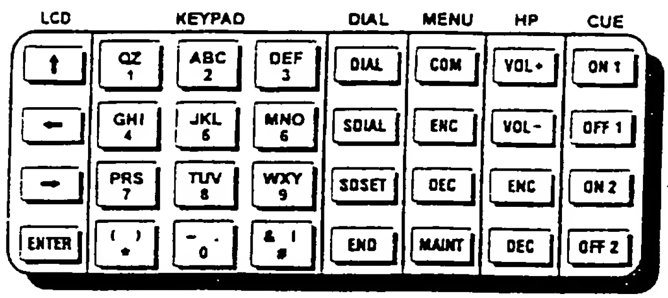


Figure 6

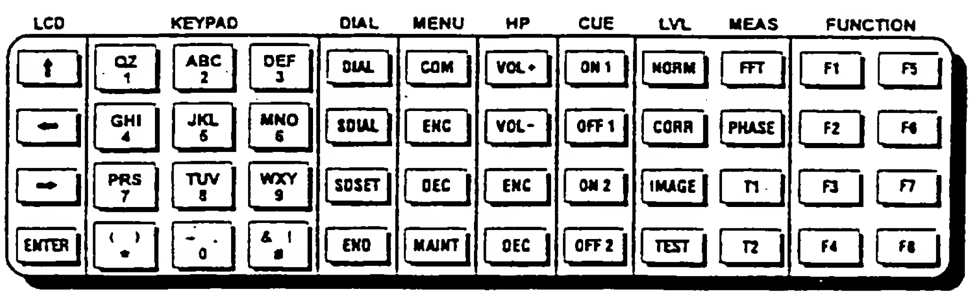


Figure 7

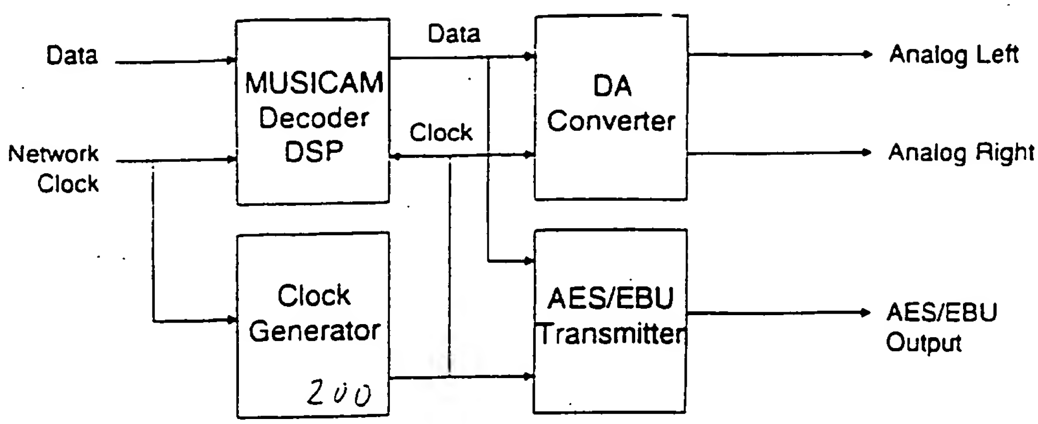


Figure 8

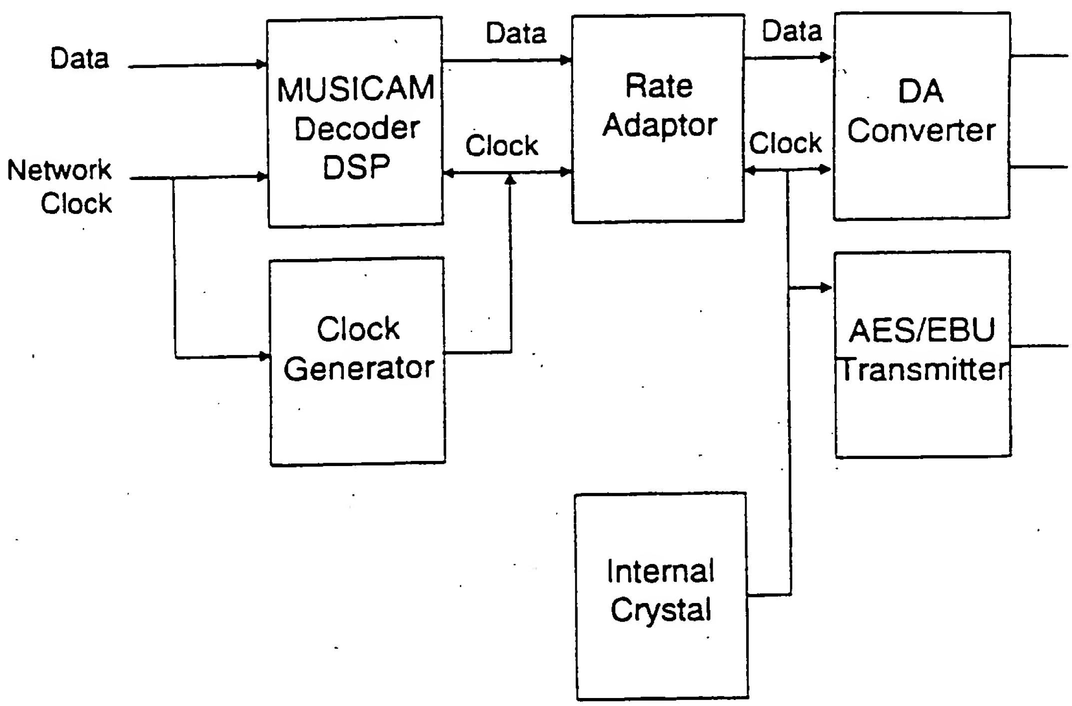


Figure 9

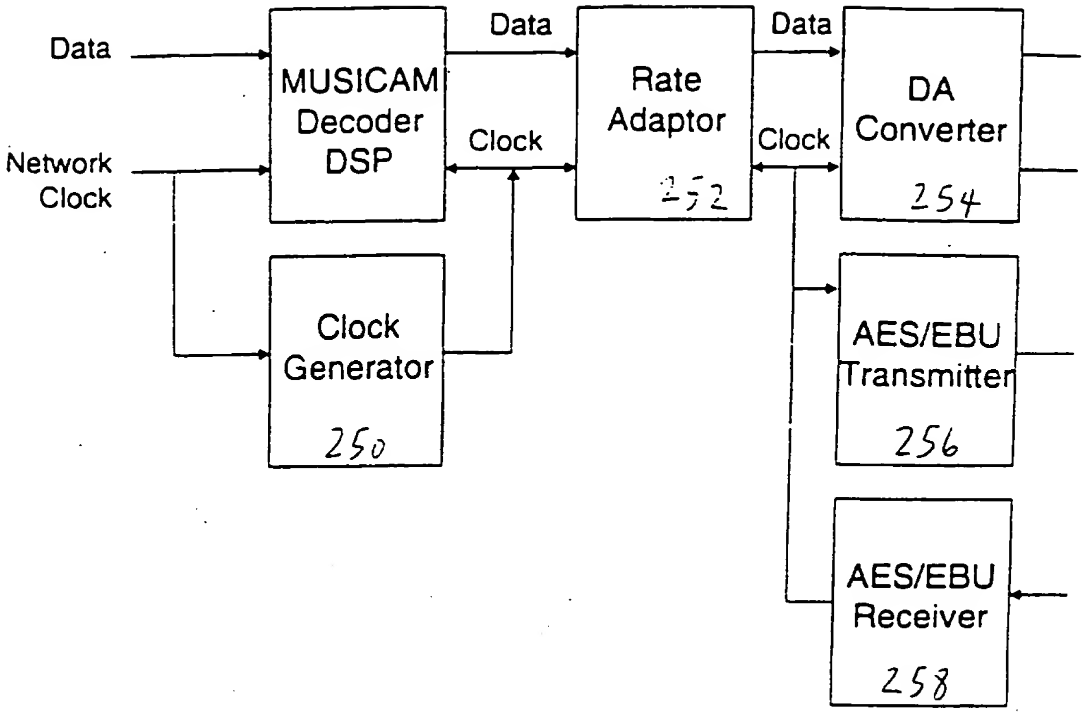

Data
Network Clock
MUSICAM Decoder DSP
Data
Clock
Clock Generator
250
Rate Adaptor
252
Data
Clock
DA Converter
254
AES/EBU Transmitter
256
AES/EBU Receiver
258

Figure 10

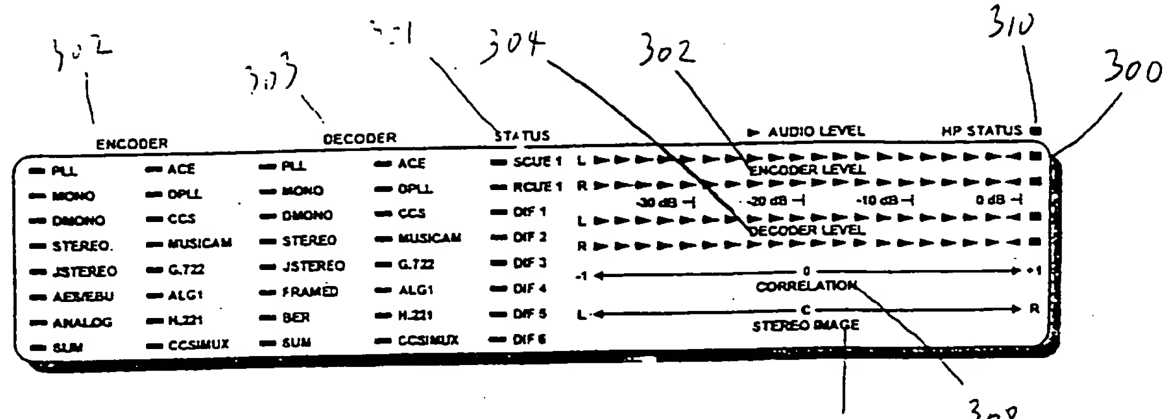


Figure 11

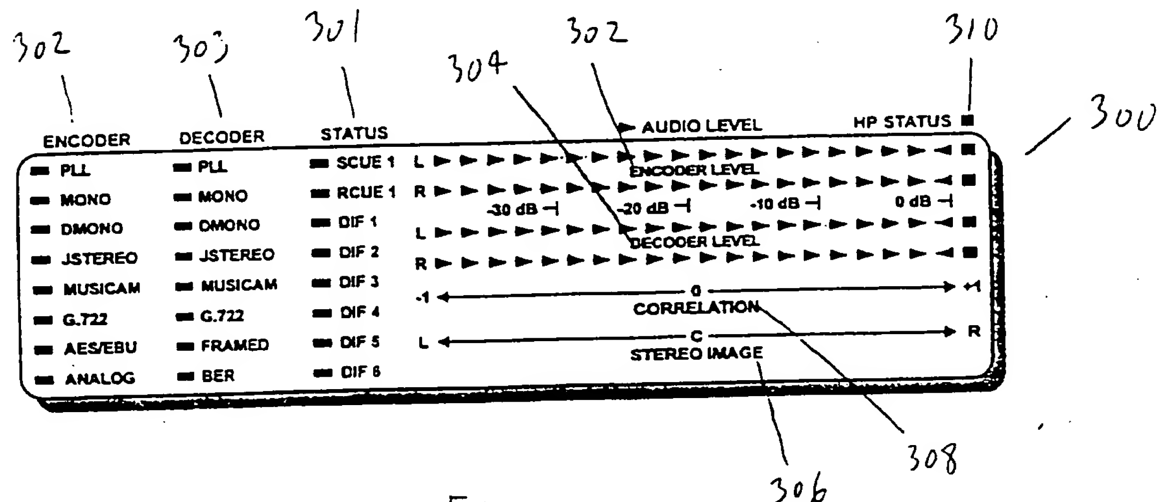


Figure 12

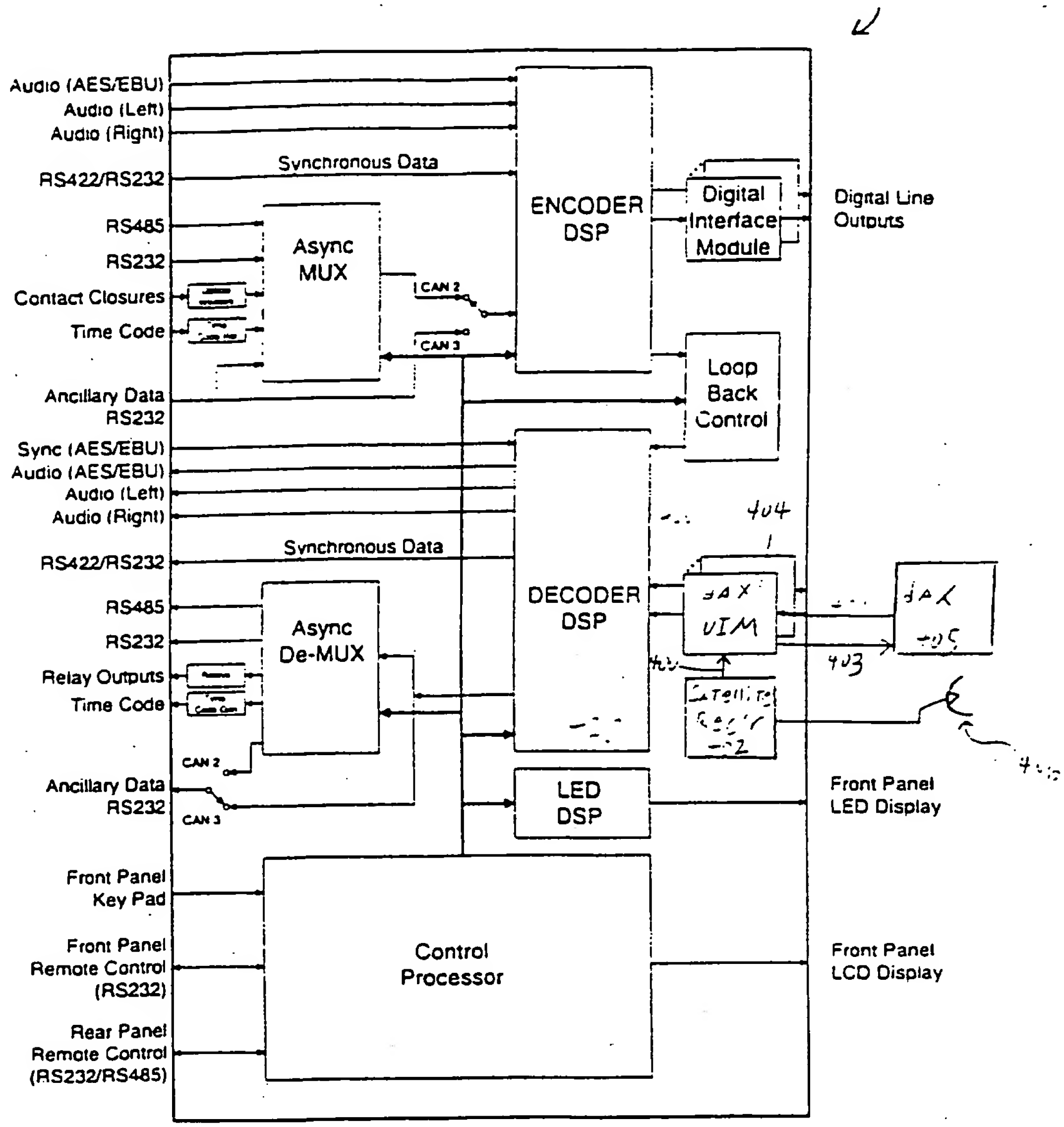

Audio (AES/EBU)
Audio (Left)
Audio (Right)
Synchronous Data
RS422/RS232
RS485
RS232
Contact Closures
Time Code
Async MUX
CAN 2
CAN 3
ENCODER DSP
Digital Interface Module
Digital Line Outputs
Loop Back Control
Ancillary Data RS232
Sync (AES/EBU)
Audio (AES/EBU)
Audio (Left)
Audio (Right)
Synchronous Data
RS422/RS232
RS485
RS232
Relay Outputs
Time Code
Async De-MUX
DECODER DSP
Ancillary Data RS232
CAN 2
CAN 3
LED DSP
Front Panel LED Display
Front Panel Key Pad
Front Panel Remote Control (RS232)
Rear Panel Remote Control (RS232/RS485)
Control Processor
Front Panel LCD Display
404
403

FIG. 14

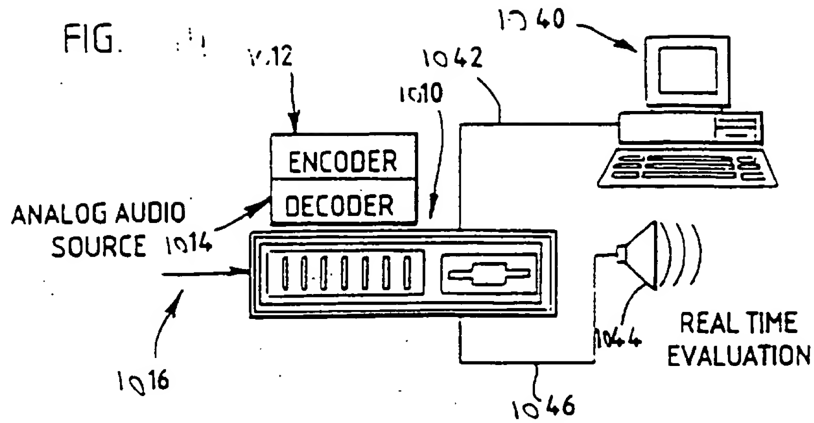

1040
1042
1012
1010
ENCODER
DECODER
ANALOG AUDIO SOURCE
1014
1016
1044
REAL TIME EVALUATION
1046

FIG. 15

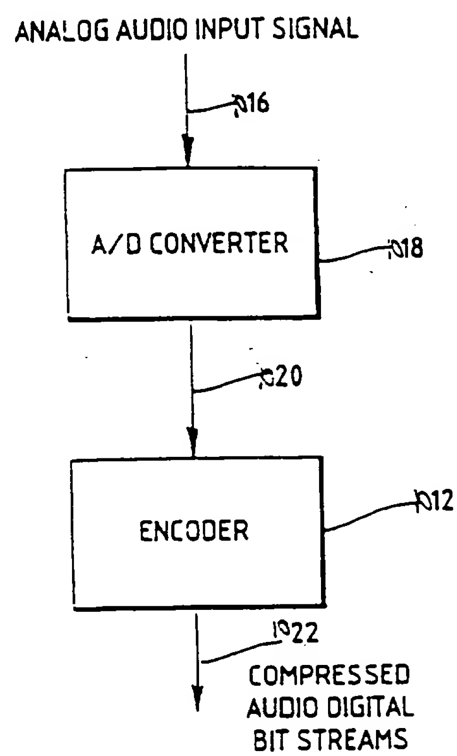

ANALOG AUDIO INPUT SIGNAL
1016
A/D CONVERTER
1018
1020
ENCODER
1012
1022
COMPRESSED AUDIO DIGITAL BIT STREAMS

FIG. 16

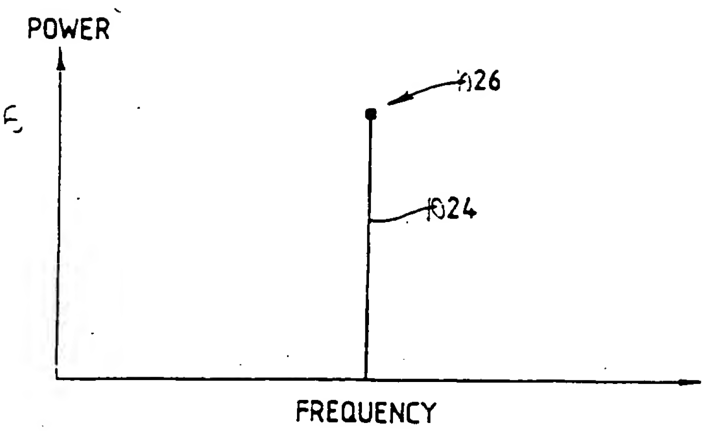


FIG 17

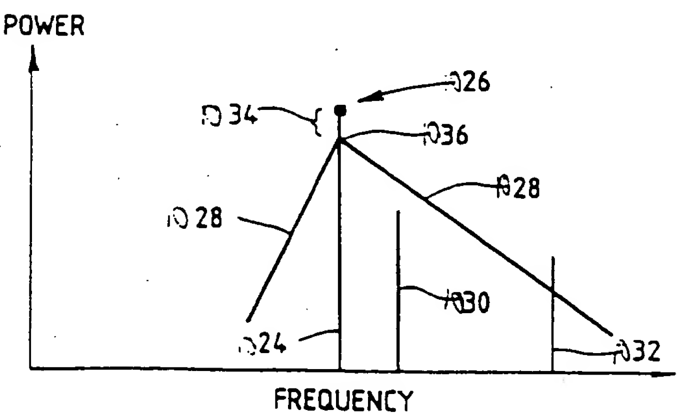


FIG. 18

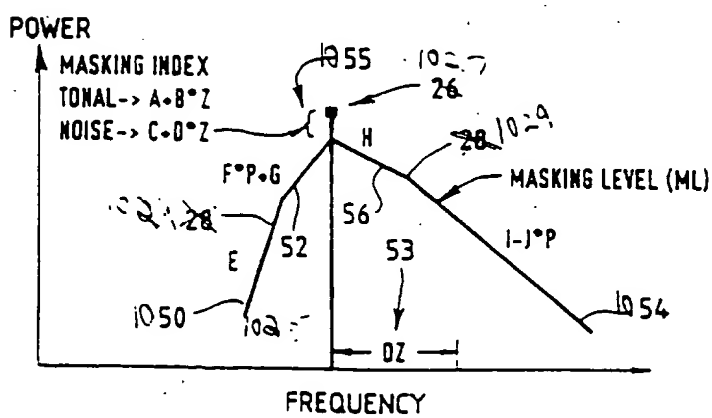


FIG. 19

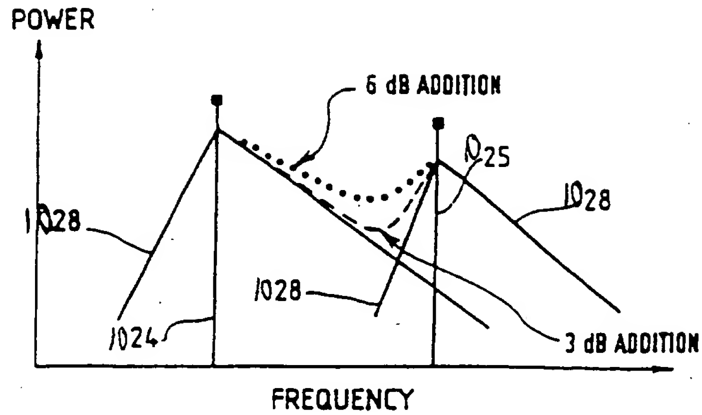
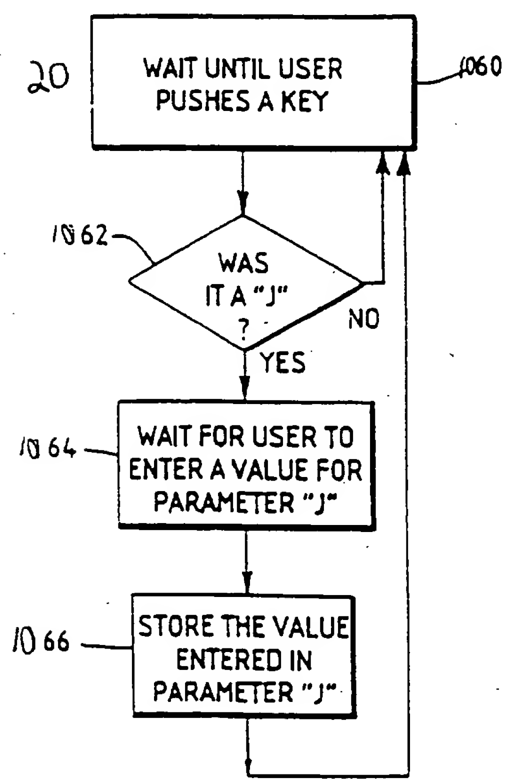
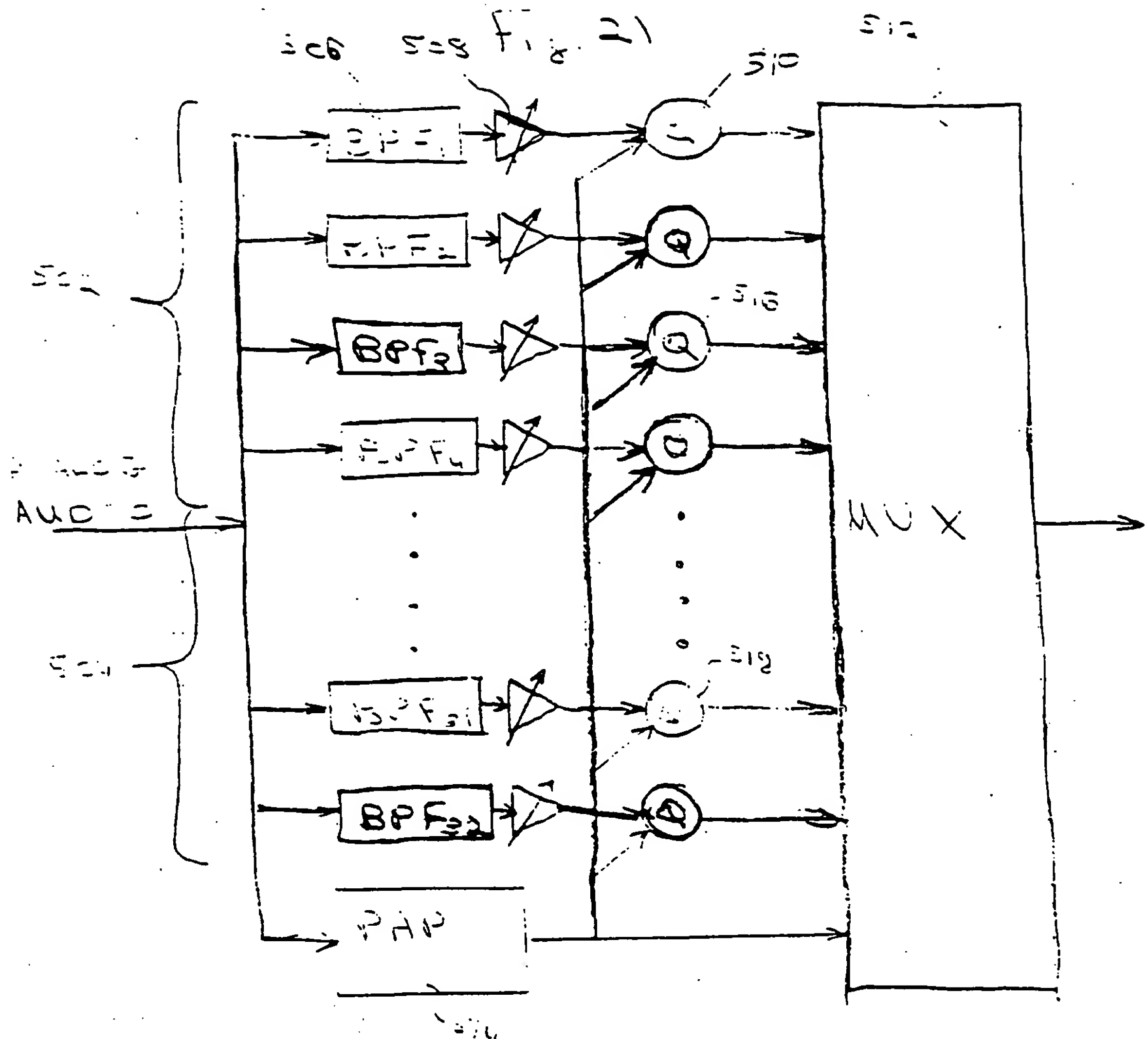
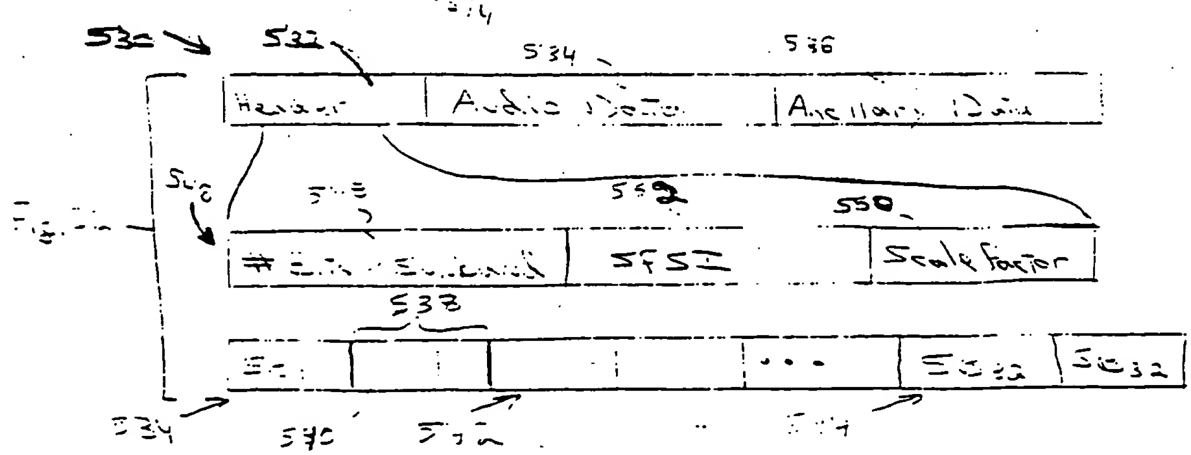

POWER
6 dB ADDITION
1025
1028
1028
1028
1024
3 dB ADDITION
FREQUENCY

FIG. 20


WAIT UNTIL USER PUSHES A KEY
1060
1062
WAS IT A "J" ?
NO
YES
1064
WAIT FOR USER TO ENTER A VALUE FOR PARAMETER "J"
1066
STORE THE VALUE ENTERED IN PARAMETER "J"


510
BPF3
BPF32
AUDIO
MUX


530
532
534
536
Header
Audio Data
Ancillary Data
550
SFSI
Scale factor
538

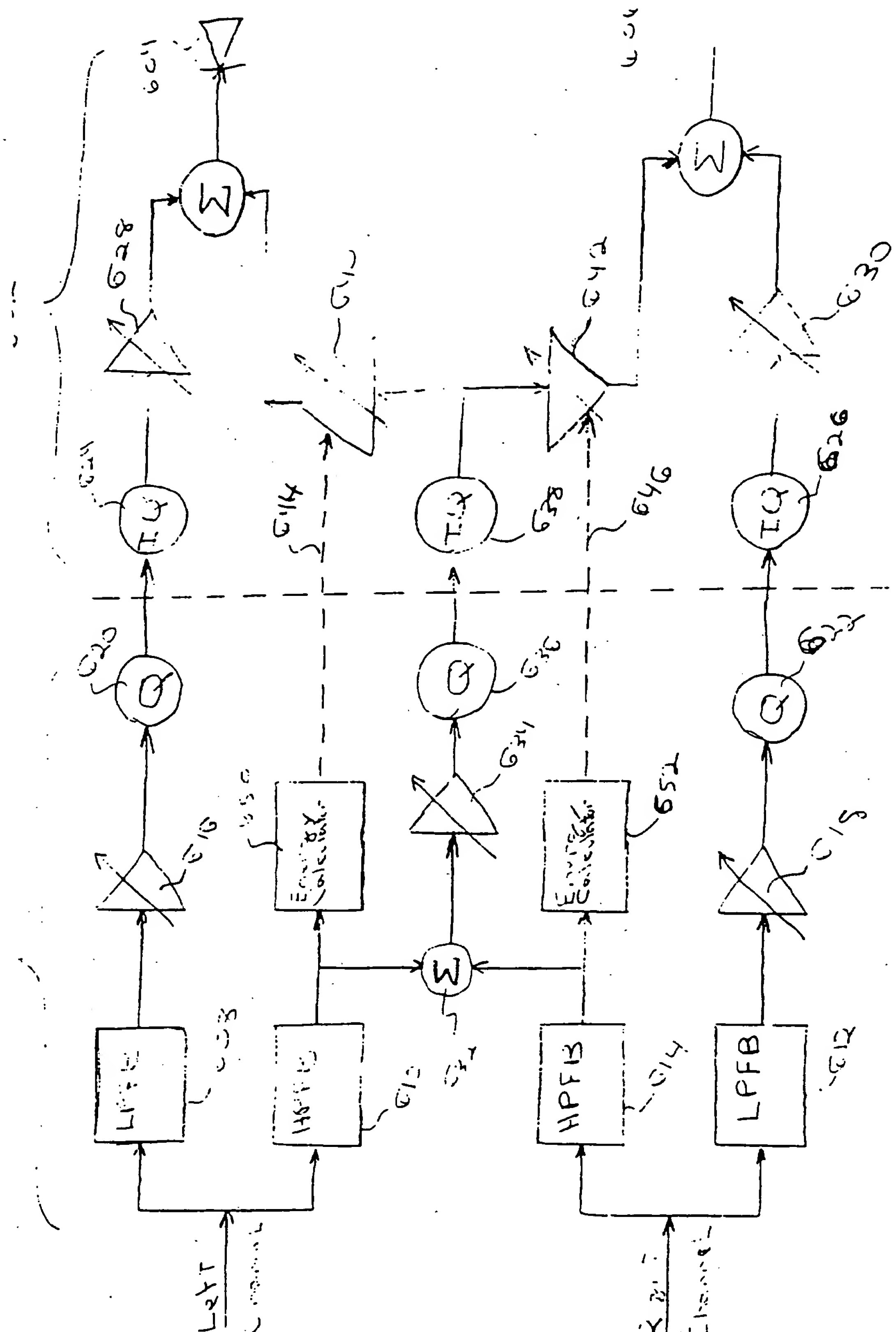

Left Channel
Right Channel
LPFB
HPFB
Energy Calculator
Q
IQ
M
612
614
616
618
620
622
626
628
630
632
634
636
638
642
646
650
652

FIG. 23

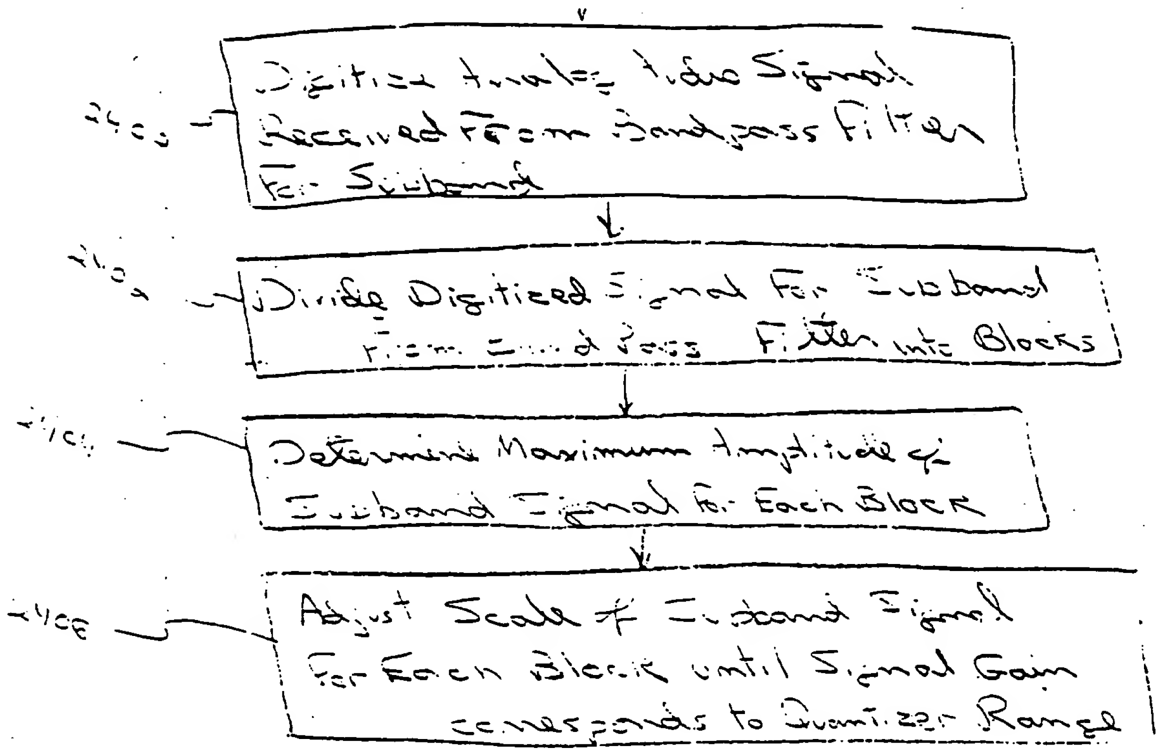

Digitize Analog Video Signal Received From Bandpass Filter For Subband
Divide Digitized Signal For Subband From Band Pass Filter into Blocks
Determine Maximum Amplitude of Subband Signal For Each Block
Adjust Scale of Subband Signal For Each Block until Signal Gain corresponds to Quantizer Range

Fig. 24

Fig. 25A

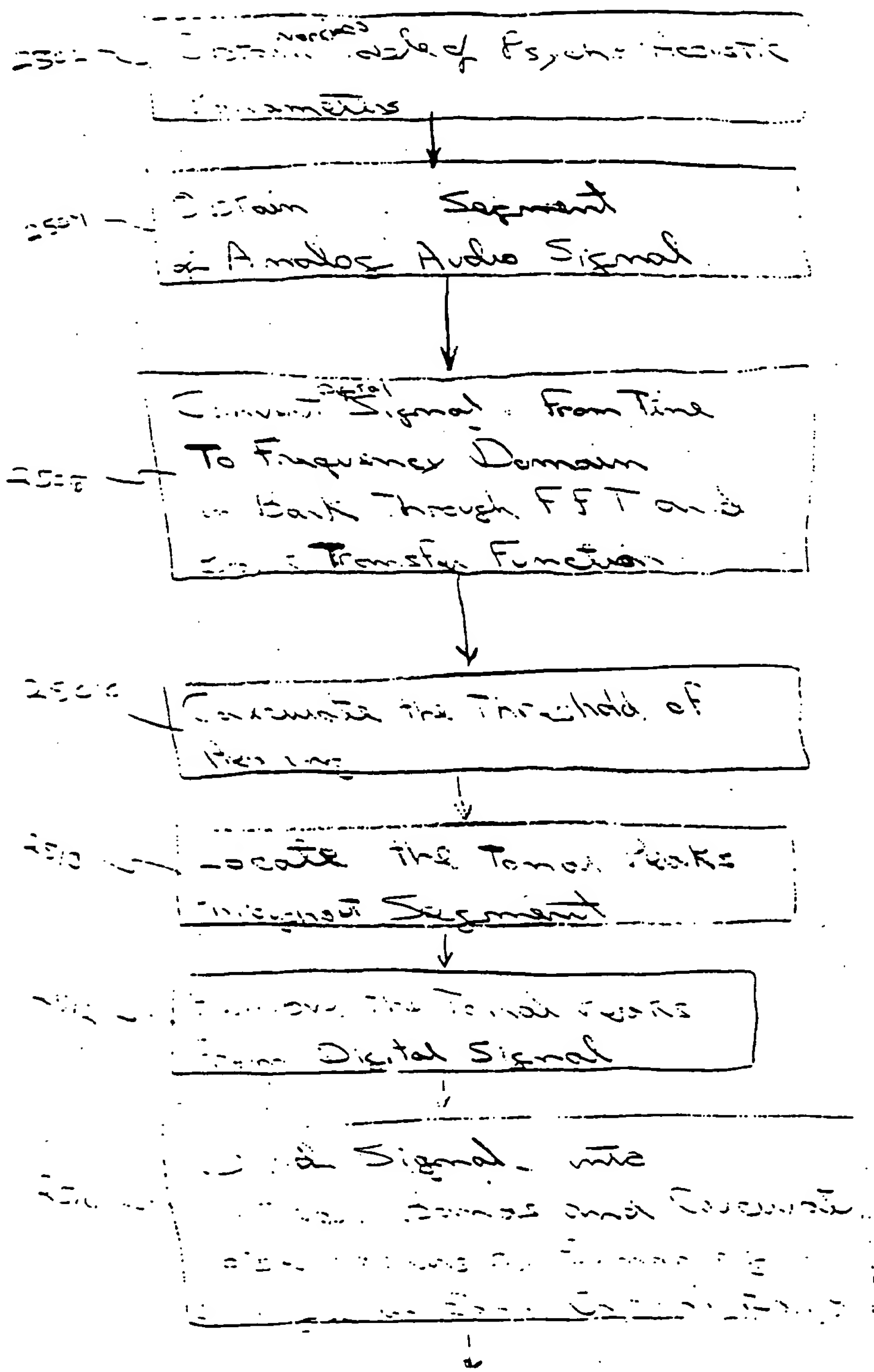
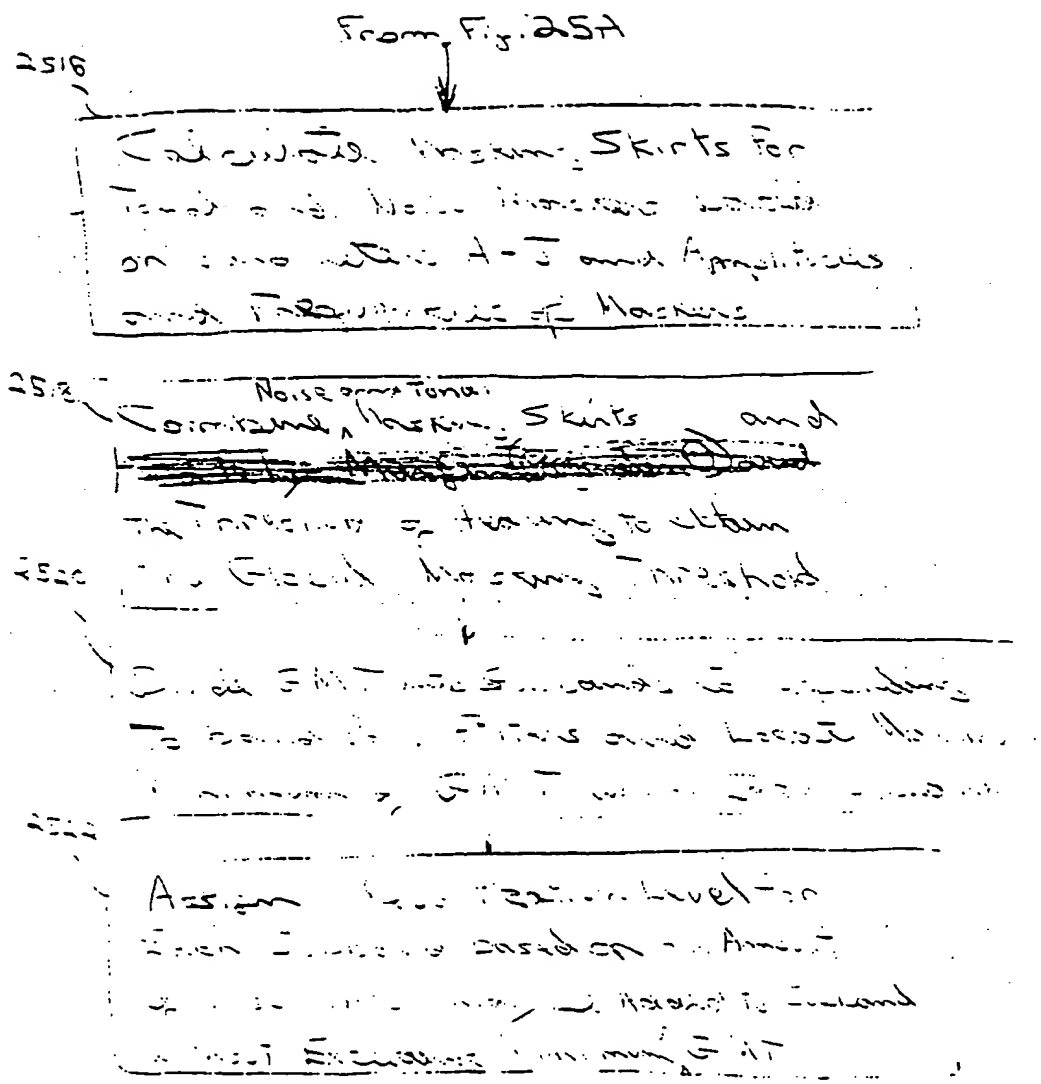
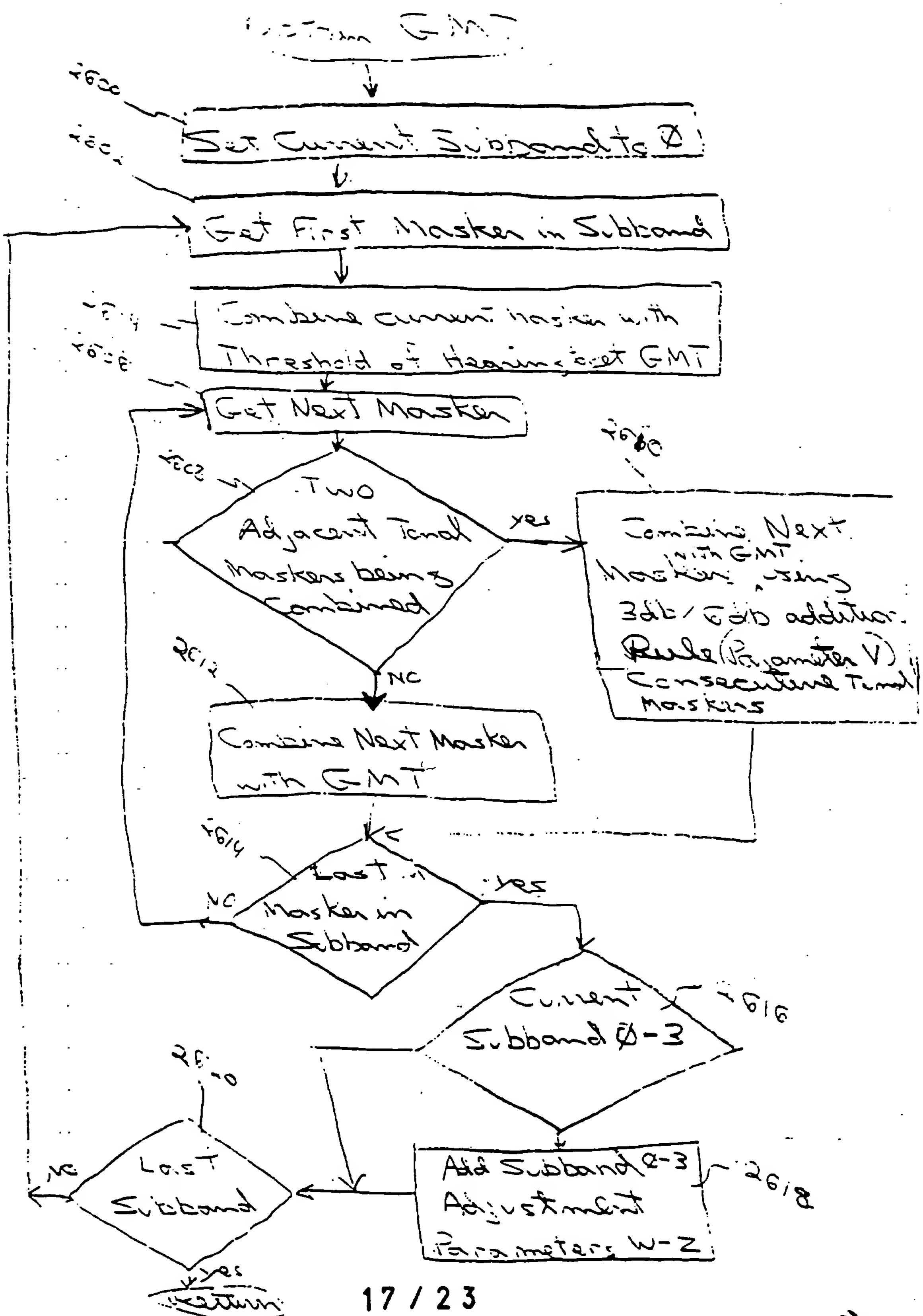

Psycho Acoustic
Segment
Analog Audio Signal
Signal From Time
To Frequency Domain
Back Through FFT
Transfer Function
Threshold of
Locate the Tonal Peaks
Segment
Digital Signal
Signal

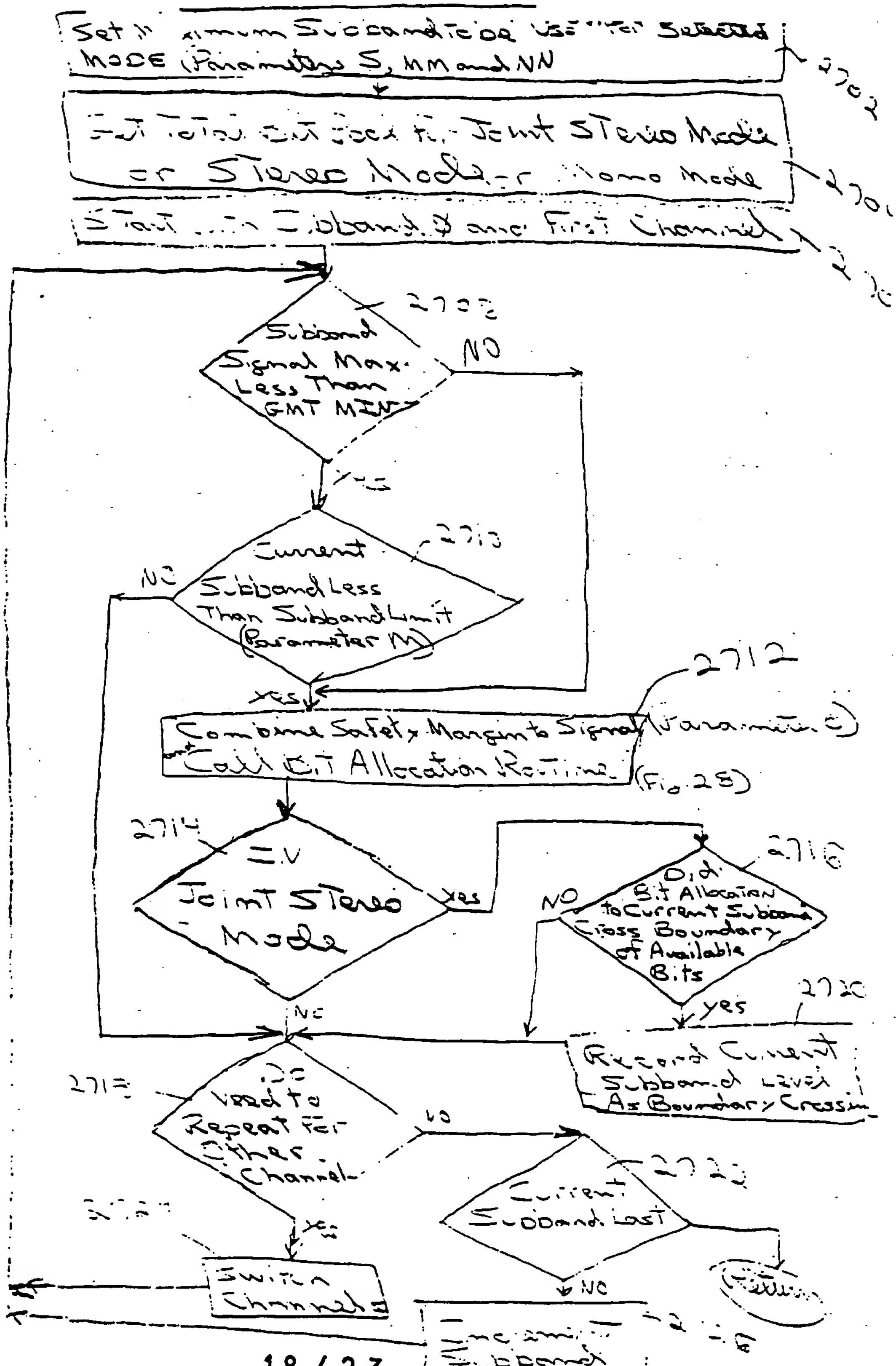
Fig. 25B

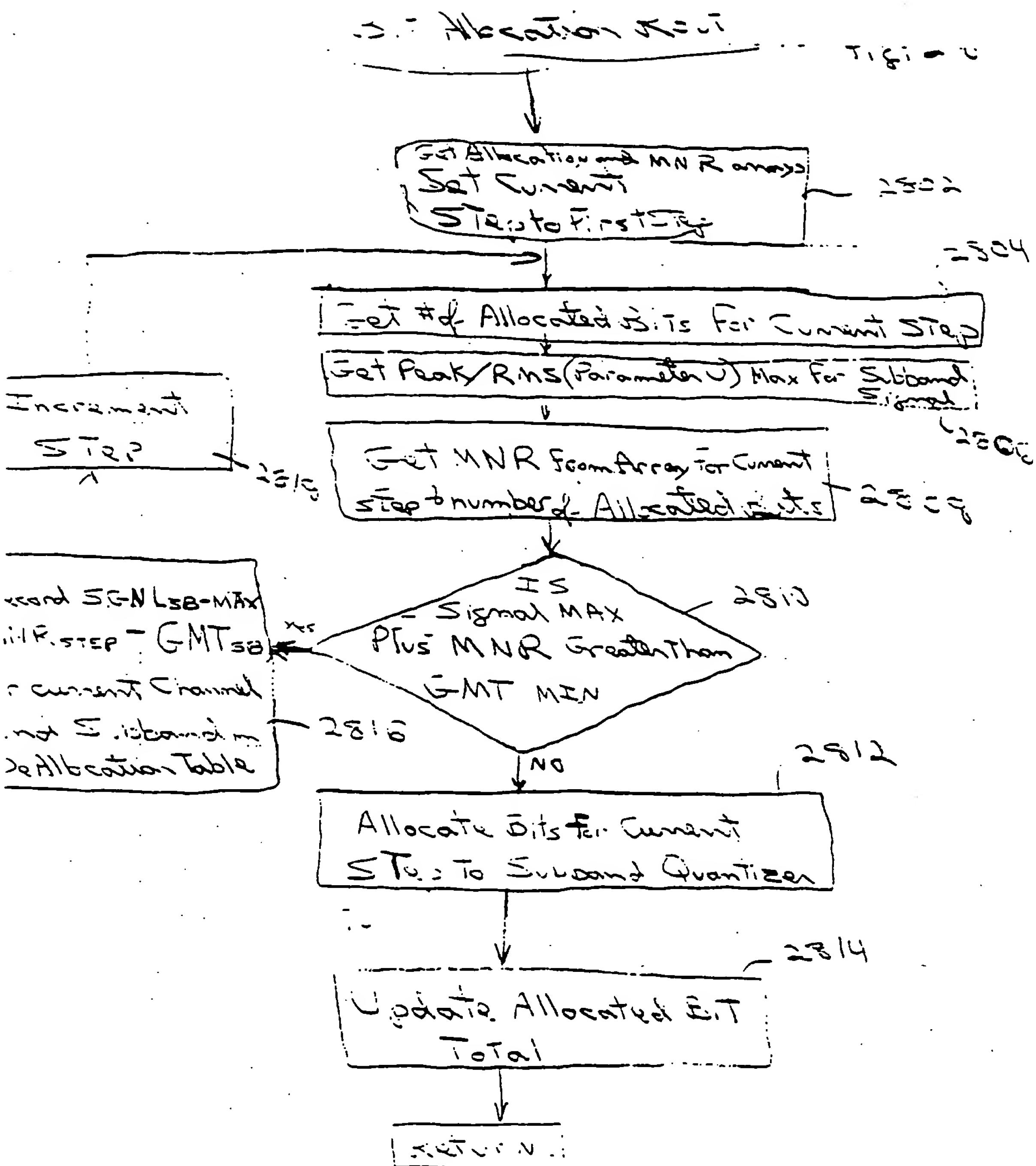




Return GMT
2600
Set Current Subband to Ø
2602
Get First Masker in Subband
2604
Combine current masker with Threshold of Hearing set GMT
2608
Get Next Masker
2602
Two Adjacent Tonal Maskers being Combined
yes
2610
Combine Next Masker using 3dB/6dB addition Rule (Parameter V) with GMT Consecutive Tonal Maskers
NO
2612
Combine Next Masker with GMT
2614
Last Masker in Subband
NO
yes
Current Subband Ø-3
2616
Add Subband Ø-3 Adjustment Parameters W-Z
2618
2620
Last Subband
NO
yes
Return

Fig. 27


MODE (Parameters S, MM and NN
Joint Stereo Mode
or Stereo Mode
First Channel
2702
2704
2706
2708
Subband
Signal Max.
Less Than
GMT MIN
NO
Yes
Current
Subband Less
Than Subband Limit
(Parameter M)
NO
2710
2712
Yes
Combine Safety Margin to Signal
Call Bit Allocation Routine
(Parameter)
(Fig. 28)
2714
Joint Stereo
Mode
Yes
NO
2716
Did
Bit Allocation
to Current Subband
Cross Boundary
of Available
Bits
NO
yes
2720
Record Current
Subband Level
As Boundary Crossing
2718
Need to
Repeat for
Other
Channel
Yes
Switch
Channel
2722
Current
Subband Last
NO
Return
2726


Get Allocation and MNR arrays
Set Current
Step to First Step
2802
2804
Get # of Allocated Bits For Current Step
Get Peak/RMS (Parameter U) Max for Subband Signal
2806
Increment
Step
2810
Get MNR from Array for Current
step to number of Allocated Bits
2808
IS
Signal MAX
Plus MNR Greater Than
GMT MIN
2810
YES
GMT SB
De Allocation Table
2816
NO
2812
Allocate Bits for Current
Step To Subband Quantizer
2814
Update Allocated Bit
Total

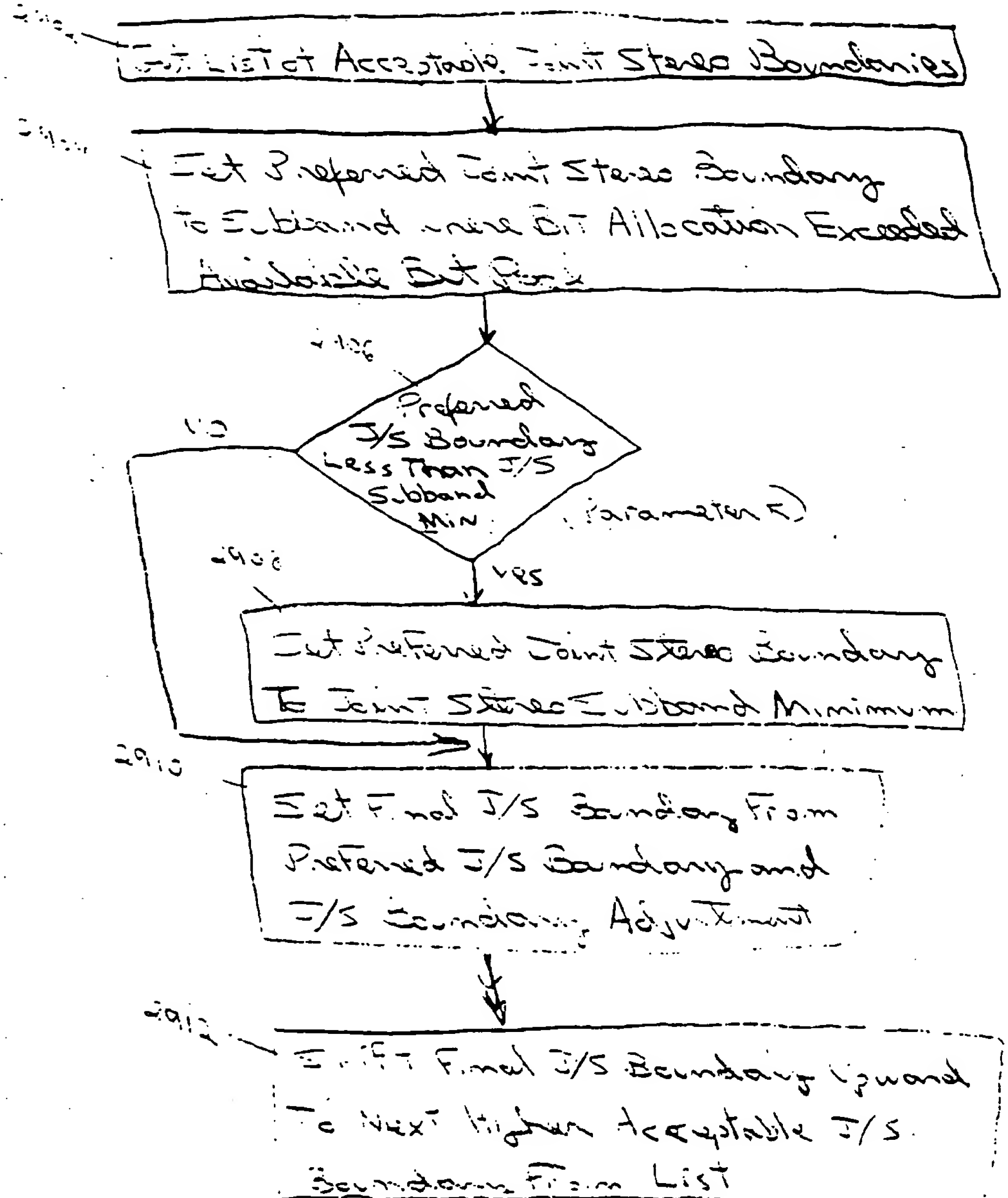

Get List of Acceptable Joint Stereo Boundaries
Set Preferred Joint Stereo Boundary To Subband where Bit Allocation Exceeded Available Bit Pool
Preferred J/S Boundary Less Than J/S Subband Min
(Parameter F)
NO
YES
Set Preferred Joint Stereo Boundary To Joint Stereo Subband Minimum
Set Final J/S Boundary From Preferred J/S Boundary and J/S Boundary Adjustment
Shift Final J/S Boundary Upward To Next Higher Acceptable J/S Boundary From List

Fig. 29

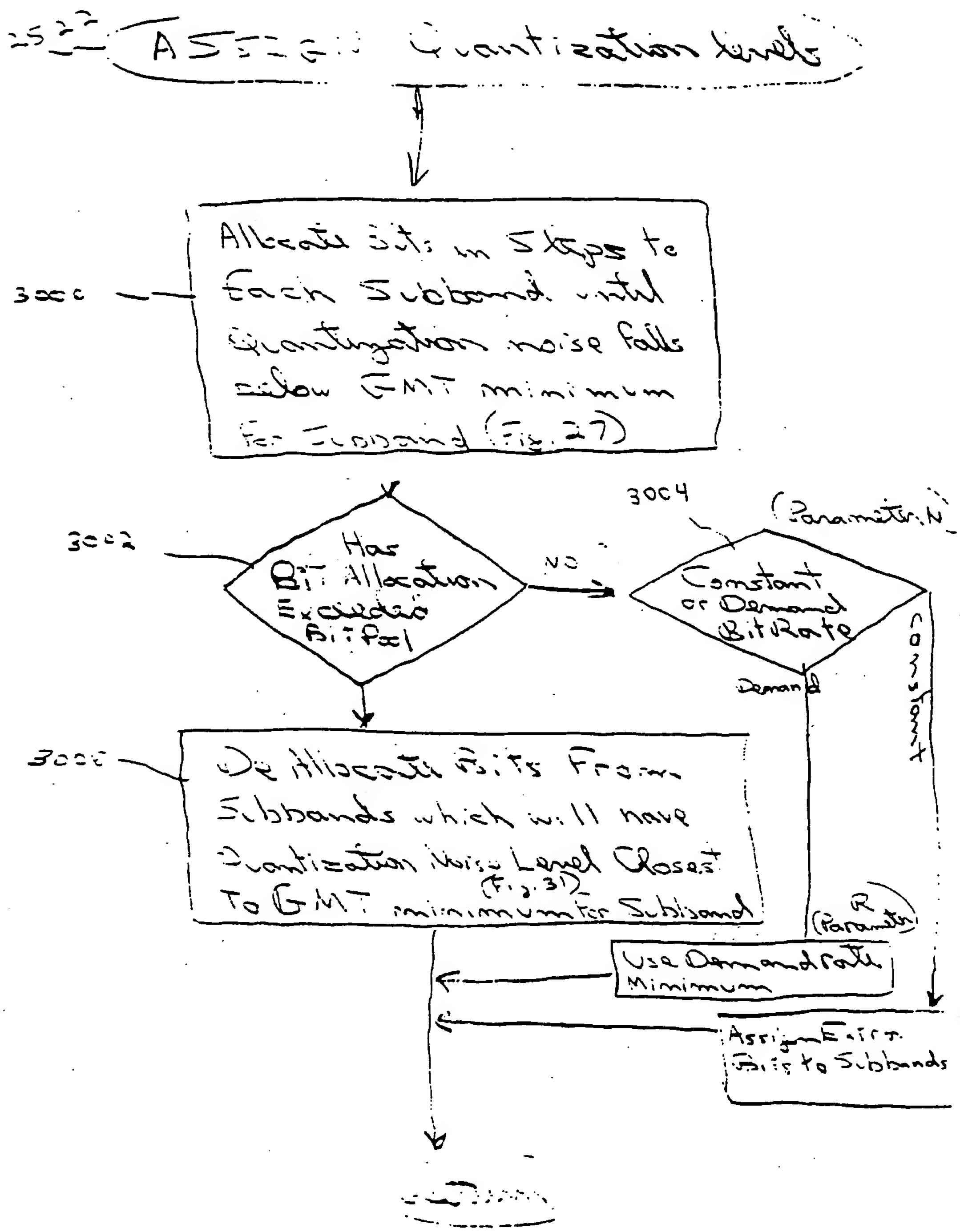
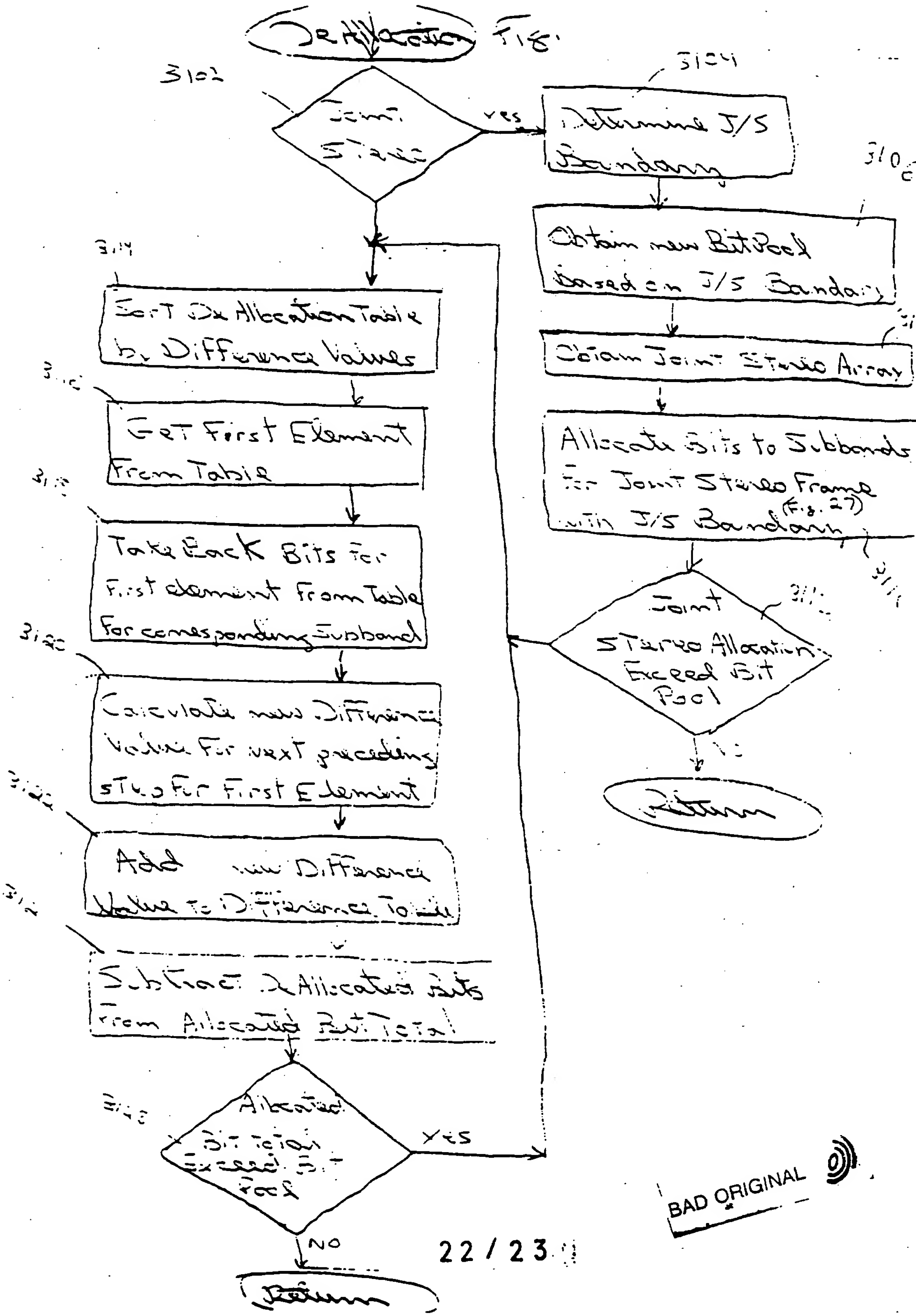

Assign Quantization levels
Allocate Bits in Steps to Each Subband until Quantization noise falls below GMT minimum for Subband (Fig. 27)
3000
3002
Has Bit Allocation Exceeded Bit Pool
No
3004
(Parameter N)
Constant or Demand Bit Rate
Demand
Constant
De Allocate Bits From Subbands which will have Quantization Noise Level Closest To GMT minimum for Subband (Fig. 31)
(Parameter R)
Use Demand rate Minimum
Assign Extra Bits to Subbands

Fig. 30


De Allocation
Joint Stereo
Yes
Determine J/S Boundary
Obtain new Bit Pool Based on J/S Boundary
Obtain Joint Stereo Array
Allocate Bits to Subbands For Joint Stereo Frame with J/S Boundary (Fig. 27)
Joint Stereo Allocation Exceed Bit Pool
No
Return
Sort De Allocation Table by Difference Values
Get First Element From Table
Take Back Bits For First element From Table For corresponding Subband
Calculate new Difference Value For next preceding step For First Element
Add new Difference Value to Difference Table
Subtract De Allocated Bits From Allocated Bit Total
Allocated Bit Total Exceed Bit Pool
Yes
No
Return

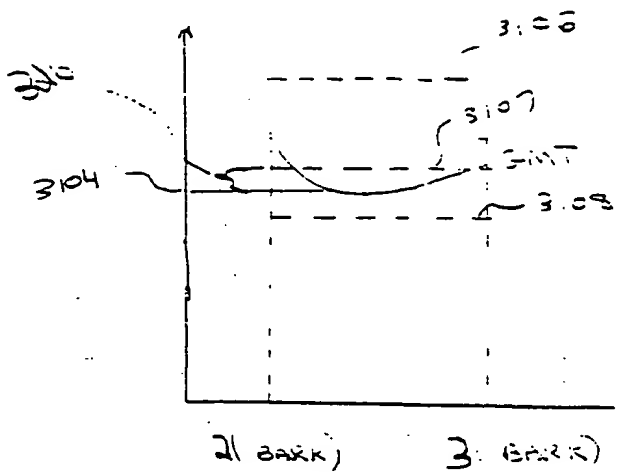


Fig. 32a

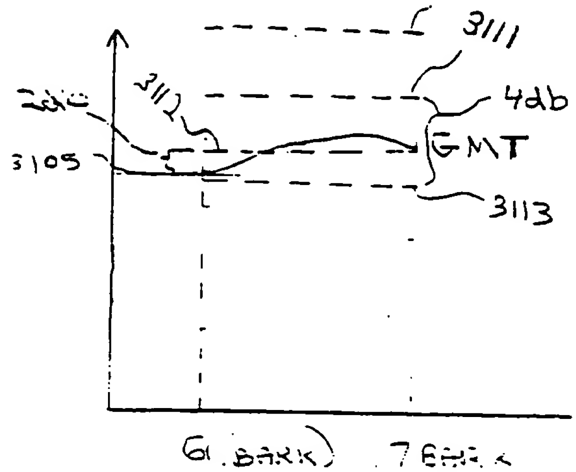


Fig. 32b

Fig. 33

Channel	Subband	Difference Value	Allocation STEP
	7	2db	3
2	3	3db	2
	7	4db	2

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/04974

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G10L 3/00
US CL :395/2.1

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/2.1, 2.12, 2.14, 2.2, 2.34, 2.38, 2.39

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS: speech or audio, ancillary, auxillary, psycho-acoustic

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y,E	US, A, 5,530,655 (LOKHOFF ET AL) 25 June 1996, Abstract, Figures 12 and 16.	1-3
X,E Y,E	US, A, 5,515,107 (CHIANG ET AL) 07 May 1996, abstract, figures 1-3, 4E.	1, 3 2
Y	US, A, 5,161,210 (DRUYVESTEYN ET AL) 03 November 1992, abstract, figure 1.	1-3
Y,P	US, A, 5,493,647 (MIYASAKA ET AL) 20 February 1996, abstract.	1-3
Y	US, A, Re32,124, (ATAL) 22 April 1986, abstract, figures 1, 2.	2

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be part of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
14 AUGUST 1996	10 SEP 1996

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	DAVID D. KNEPPER Telephone No. (703) 305-9600